WEST

Generate Collection | Print

Search Results - Record(s) 1 through 23 of 23 returned.

☐ 1. Document ID: US 6526351 B2

L3: Entry 1 of 23 File: USPT Feb 25, 2003

DOCUMENT-IDENTIFIER: US 6526351 B2
TITLE: Interactive multimedia tour guide

Detailed Description Text (2):
Referring now to the drawings, and more particularly to FIG. 1, there is shown a computer system architecture on which the interactive multimedia tour guide may be installed for the purpose of previewing, selecting and customizing tours. The computer system includes a central processor unit (CPU) 100 connected to a memory controller hub (MCH) 101 via a system bus 102. The MCH 101 is connected to a random access memory (RAM) 103 via a direct memory bus 104, such as a bus meeting the Rambus specification, and a video graphics chip 105, such as the accelerated graphics port (AGP) graphics chip, via a video bus 106. A display screen 140, such as a liquid crystal display (LCD), is connected to the AGP graphics chip 105.

Detailed Description Text (22):
The system unit 53 is shown in more detail in FIG. 6. The system unit includes a CPU 601 connected by a back-side bus 602 to a Level 2 cache RAM 603. The CPU 601 is also connected by a front-side bus 604 to a direct memory access (DMA) control chip 605 to which to which main system RAM 606 is connected by memory bus 607 and AGP video controller 608 is connected by AGP bus 609. The AGP video controller is connected to a video bus 610 which connects to the display 52 via a video frame buffer 611. The DMA control chip 605 is connected by a PCI (Peripheral Component Interconnect) bus 612 to an I/O control chip 613. The I/O control chip 613 supports an internal hard drive 614, a CD/DVD drive 615 and various ports, including an IEEE 1394 port 616, one or more USB ports 617, an IR port 618, and a wireless RF port 619, such as for the Bluetooth standard. One or the other of these ports may be used to provide a connection to the user's PC (FIG. 1) to download a selected tour to the hard drive 614 or a connection to the optional digital camera 57 for accessing the GIS database on hard drive 614 and/or CD/DVD drive 615 in order to identify subject(s) of digital video image(s) or video clip(s). Operating system (OS) software is installed on the hard drive 614. Voice recognition and speech synthesis software are also installed on the hard drive 614. The I/O control chip 613 additionally provides support for a modem 620 which connects to the wireless mobile telephone 44 and an audio chip 621 that connects to a microphone 623 and a speaker 624. The microphone 623 could be integrated into the display 52, and the speaker 624 could be integrated into the system unit 53. An audio jack 625 is provided in the system unit 53 in order to plug the optional headset 58 into the system unit. The jack 625 disables the integrated microphone and speaker when the headset 57 is plugged into the system unit. Optionally, a low power RE transmitter 626 may be used to transmit to a radio in an automobile in order to use the automobile's speaker system. The controls 522 to 526 on the display 52 are also connected to the I/O control chip 613. These controls 522 to 526 provide only minimal control functions since most of the computer/user interface is supported by the voice recognition software and speech synthesis software. The display 52 could additionally incorporate functions of a personal digital assistant (PDA) in which a stylus is used to access an address book, a reminder list, a calculator and the like from one or menus and to input written data using the handwriting recognition function of the PDA function. As mentioned, the display 52 could be the PDA 40 shown in FIG. 4.

☐ 2. Document ID: US 6523092 B1

L3: Entry 2 of 23 File: USPT Feb 18, 2003

DOCUMENT-IDENTIFIER: US 6523092 B1
TITLE: Cache line replacement policy enhancement to avoid memory page thrashing

Abstract Text (1):
A method for a cache line replacement policy enhancement to avoid memory page thrashing. The method of one embodiment comprises comparing a memory request address with cache tags to determine if any cache entry in set `n` can match the address. The address is masked to determine if a thrash condition exists. Allocation to set `n` is discouraged if a thrash condition is present.

Brief Summary Text (2):
The present invention relates generally to the field of computers and computer systems. More particularly, the present invention relates to a cache line replacement policy to avoid memory page thrashing.

Brief Summary Text (5):
Computer systems have evolved to include memory hierarchies comprising various types of long term storage, main memory, and caches. However, as one moves down the down the memory hierarchy from caches to long term storage, device access times increase dramatically. An ideal solution is to have enough cache memory or fast main memory available to service the currently executing program. But in most systems, such memory is present in only limited amounts or the program demands more memory than is available.

Brief Summary Text (6):
Caches are generally used to keep often used or recently used data close to or within the processor. The idea is that by storing recently used data in close proximity to the processor, the next time a memory request is made for that particular data, a long memory access to main memory or the hard disk drive is not necessary. When a computer starts up, the cache is empty. But over time, the cache continues to fill up until there are no longer any empty entries for new data. This is not a problem as long as invalid entries are available for replacement. But if all existing entries are valid, the cache replacement logic must delete valid entries to make room for incoming data.

Brief Summary Text (8):
Some of these conflicts are inevitable, as operating system, application, and chipset design details can interact in ways to create vicious cases. On the other hand, a substantial class of conflicts can arise simply from an interaction between the way in which memory controllers are designed and the way in which processor caches are designed. These conflicts can have a significant impact on processor and overall system performance.

Drawing Description Text (3):
FIG. 1 is a block diagram of one embodiment of a system employing a cache line replacement policy enhancement to avoid memory page thrashing;

Drawing Description Text (4):
FIG. 2 is a block diagram of a cache memory to employ one embodiment of the present invention;

Drawing Description Text (6):
FIG. 3B illustrates a four-way set associative 128 Kbyte cache;

Detailed Description Text (2):
A method and apparatus for a cache line replacement policy enhancement to avoid memory page thrashing is disclosed. The embodiments described herein are described in the context of a microprocessor, but are not so limited. Although the following embodiments are described with reference to processors, other embodiments are applicable to other integrated circuits or logic devices. The same techniques and teachings of the present invention can easily be applied to other types of circuits or semiconductor devices that use caches or memory.

Detailed Description Text (4):
Referring now to FIG. 1, a computer system 100 is shown. System 100 includes a component, such as a memory controller hub 116, employing a cache replacement mechanism 106 in accordance with the present invention, such as in the embodiment described herein. System 100 is representative of processing systems based on the Intel PENTIUM.RTM. Pro, PENTIUM II, PENTIUM III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 100 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (6):
System 100 includes a memory 120. Memory 120 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102. A cache memory 104 can reside inside processor 102 that stores data signals stored in memory 120. Alternatively, in another embodiment, the cache memory such as external cache 108 can reside external to the processor.

Detailed Description Text (7):
A system logic chip 116 is coupled to the processor bus 110 and memory 120. The system logic chip 116 in the illustrated embodiment is a memory controller hub (MCH). The processor 102 communicates to the MCH 116 via a processor bus 110. The MCH 116 provides a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 116 directs data signals between processor 102, memory 120, and other components in the system 100 and bridges the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 provides a graphics port for coupling to a graphics controller 112. The MCH 116 is coupled to memory 120 through a memory interface 118. The graphics card 112 is coupled to the MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

Detailed Description Text (8):
A cache replacement mechanism 106 to avoid memory page thrashing also resides in MCH 116. Alternate embodiments of a cache replacement mechanism 106 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits.

Detailed Description Text (11):
Processor caches most often are arranged so that the lower bits of a request address are used to index into the memory array and the upper bits are compared against a cache tag to determine the relevance of the data in the cache to the request being offered. As a result, requests whose addresses differ only in these upper bits are assigned to the same line of the cache array, since the lower bits are used as the index. Because this behavior can cause inefficient use of the cache, many caches are divided into a number of smaller sets. These sets diffuse the effect of pathological address patterns by creating several locations where a given request can be stored, thus dramatically reducing the probability of an addressing pathology. In such multi-set (or multi-way) caching implementations, requests whose addresses miss the cache are typically assigned to a cache set on a least-recently-used, least-recently-allocated, or random basis.

Detailed Description Text (12):
One form of a cache that is often used is known as a write-back cache. In a writeback cache implementation, write operations offered by the processor core are retained in the cache and written back to memory only when the cache line needs to be recycled in order to service another cache transaction.

Detailed Description Text (13):
FIG. 2 is a block diagram of a cache memory 200 to employ one embodiment of the present invention. The cache memory 200 of FIG. 2 is four-way set associative with `n` sets. Each of the `n` sets consists of four separate entries. Cache 200 comprises of Bank 0 210, Bank 1 220, Bank 2 230, and Bank 3 240. Each bank provides a block to store an entry for each set. For example, Bank 0 210 stores Entry 0 212, Bank 1 220 stores Entry 1 222, Bank 2 240 stores Entry 2 242, and Bank 3 260 stores Entry 3 262.

Detailed Description Text (14):
When a memory access occurs, the cache logic receives a memory address 270. The address 270 is divided into two fields: tag 272 and block offset 276. The value in the tag field 272 is used to compare with the tags in a set when searching for a match. The block offset field 276 is used to select the desired data from the entry.

Detailed Description Text (15):
Each cache entry 260 comprises of three fields: valid 262, tag 264, and data 266. The valid field 262 indicates whether the present entry is currently in use or not. The tag field 264 contains part of the address 270 and is used during a cache search to determine whether an entry is the one desired. The data field 266 contains the data for this entry.

Detailed Description Text (16):
During a memory access, the cache logic parses address 270 into the separate fields 272 and 276. Each tag 272 is compared with the tag 264 of each valid entry in all sets of the cache. If a match is found, then the data 266 is sent to the processor. But if no match is found, then the cache 200 does not have the desired data. After the data is retrieved from main memory, the data will also be stored into the cache 200 for future reference. If one or more of the cache sets contains an invalid/empty entry at the location pointed to by the block offset 276, then the data is simply stored in that set. But if all the sets have valid data at the location indicated by the block offset 276, then the cache replacement policy determines which of the entries to discard. Two strategies are that are often used in cache replacement algorithms are: (1) random and (2) least recently used (LRU). To spread allocation uniformly, candidate entries are randomly selected in the random strategy. The LRU strategy attempts to reduce the chance of throwing out information that will be needed soon and records the accesses made to each entry. The set selected for replacement is the one that contains data that has been unused for the longest time.

Detailed Description Text (17):
However, if the data at a given block offset in all of the sets is repeatedly used, then frequent replacements may happen. For instance, in the four-way set associate of this example, there are four cache sets. But if there are more than four addresses that are repeatedly accessed and the data are all stored at the same block offset, then the cache will continually swap the entries of that block offset in and out.

Detailed Description Text (24):
When a pathological pattern, such as a stream of writes or the data accesses is applied to a processor cache and chipset memory controller combination, the regularity of the accesses combined with the LRU line allocation policy can cause worst case memory behavior. Many of these patterns can be detected and defeated through a modification to the cache's LRU policy. One such modification detects likely page-thrashing conditions between reads and writeback data, and modifies the line allocation policy in response.

Detailed Description Text (25):
For example, take the case of a 128 Kbyte processor cache that is divided into four sets. FIG. 3B illustrates a four-way set associative 128 Kbyte cache. Each set can store 32 Kbyte of data. Each cache line is in this case 32 bytes wide, although other line sizes are possible in other embodiments. If the cache is empty and clean at initialization, the application of a 256 Kbyte write stream to the cache would first cause 128 Kbyte of read/invalidate operations. These read/invalidate or read-for-ownership (RFO) operations serve to acquire ownership of the cache lines for the requesting agent. After these first RFO operations, however, an undesirable behavior would begin in the cache. The accesses that attempt to write to the cache beginning at 128 Kbyte would cause an RFO operation from system address 128K. This RFO operation would be followed by a writeback operation to system address 0.

Detailed Description Text (26):
In a minimal SDRAM system that employs four 64 megabit SDRAM devices, addresses could map into the memory devices in such a way that accesses to pages separated by 64 Kbyte would address the same memory bank. As a result, both accesses, the write to 128K and the writeback at 0, would be to the same memory device. In fact, because of the system address-to-device bank mapping discussed earlier, both accesses would be to the same bank. This causes a page miss for the write operation. The memory controller needs to close the memory page that is currently open in that bank and then open up the 128K page. The next system transaction to be presented is likely be an RFO operation from 128 k+32 byte. This would again cause a page miss to occur because the last address was

to the same device in the same bank. These page misses would continue through the entire 128 Kbyte of the cache, causing worst-case behavior the whole time. Note that for this example, the read data for the 128K address is destined for set 0 in the cache because of the LRU algorithm. Note also that the dirty data in Sets 1 and 3 would not conflict with the address stream. If the page thrash condition can be detected before the processor decides which set's line should be replaced, then the pathological behavior described can be avoided by the cache controller if the controller allocates the replacement line from Set 1 or Set 3 instead. There are no interesting boundary cases where all lines would cause page conflicts since correctness is not compromised by the failure of this mechanism. If all the sets have conflicts and are discouraged, then the cache controller can simply revert to the default cache entry replacement scheme such as LRU.

Detailed Description Text (27):
In order to detect the thrash condition, the cache controller requires information about the paging behavior of the memory subsystem. Sometimes this information can be complex, as in the case where several different memory technologies are populated in the same platform. The present invention provides a mask that the cache controller can use to drive its decisions about page conflicts.

Detailed Description Text (28):
FIG. 4 is a schematic diagram of a thrash condition detector. Cache controller 400 receives input signals THRASH BIT MASK 402 and REQUEST ADDRESS 404. THRASH BIT MASK 402 is a set of address mask signals coupled to AND gate 410 to detect the thrash condition. The mask 402 ensures that only the necessary bits of the addresses are tested. The particular configuration of the mask can be affected by the memory device type, device size, and bank size. REQUEST ADDRESS 404 is the address of the current memory access. SET `n` TAGS 406 are the cache tags of Set `n` to which the new data is to be stored. For this example, REQUEST ADDRESS 404 and each tag of SET `n` TAGS 406 both comprise of thirty two bits each.

Detailed Description Text (29):
Exclusive OR gates 408 receives REQUEST ADDRESS 404 and SET `n` TAGS. Exclusive OR gates 408 compares the inputs and generates a hit or miss result 409. The exclusive OR result 409 is gated by the THRASH BIT MASK 402 at AND gate 410. The detection result of the mask bits 402 and the tag comparison 409 is delivered at the AND gate 410 output as the DISCOURAGE `n` ALLOCATION signal 412. If the DISCOURAGE `n` ALLOCATION signal 412 is active, meaning that a page conflict may occur in the present set, then the cache controller 400 can attempt to allocate the replacement line from another set instead.

Detailed Description Text (30):
Embodiments of this mask can be used in the controller to determine whether a dirty entry is likely to conflict with the current request. If a possible conflict is detected and another set has a clean entry or a non-conflicting dirty entry, then that other set's entry is allocated instead. For the example above, the mask would consist of the bits that "bound" the conflicting addresses' ranges. Since accesses separated by 64 Kbyte cause page conflicts, but 32 Kbyte separations do not, bit 16 of the mask should be set while bit 15 remains clear. The mask starts with bit 0 as the right most bit. Furthermore, since a row of SDRAM implemented with four 64 megabit devices provides 32 megabytes of storage, accesses to addresses above 32 megabyte would be to a different device and would necessarily not conflict with accesses to this device. The bit mask derived from these conditions needs to include bit 16, bit 24, and all bits in between in order for the cache controller to have enough information to avoid page conflicts. Therefore, the mask for this example would be 0x01FF0000 in hex.

Detailed Description Text (32):
The mask can be stored in the system BIOS or chipset and loaded into the processor during system startup. If the processor reads this mask at startup and stores the mask internally, then the mask can be provided to the cache controllers to filter cache request addresses and to discourage allocation of a line on a case-by-case basis.

Detailed Description Text (34):
FIG. 5 is a flow diagram for the method of one embodiment. At step 502, the thrash mask is loaded into the cache controller. The address of the memory request is also received at the controller at step 504. The tags for the set `n` that can fulfill the memory request are received at the controller at step 506. At step 508, the set `n` tags are compared with the memory request address to determine if there is a cache hit. If there is a miss, the address is masked at step 510 to determine if a conflict can happen. If the controller at step 512 determines that a conflict will not occur, then the check is

complete. But if a conflict is possible, then allocation to that set is discouraged at step 514. The process continues indefinitely every time a new memory address is received at step 504.

CLAIMS:

1. A method comprising: comparing a memory request address with cache tags to determine if any cache entry in set `n` can match said address; masking said address to determine if a thrash condition exists; and discouraging allocation to set `n` if a thrash condition is present.

4. The method of claim 1 further comprising fulfilling said memory request with a cache entry from a set other than set `n`.

5. The method of claim 1 further comprising defaulting to a normal cache replacement scheme if said thrashing condition does not exist.

6. The method of claim 1 further comprising defaulting to a normal cache replacement scheme if all cache sets are discouraged.

7. The method of claim 5 wherein said normal cache replacement scheme is a least recently used scheme.

8. The method of claim 5 wherein said normal cache replacement scheme is a least recently allocated scheme.

9. The method of claim 5 wherein said normal cache replacement scheme is a random replacement scheme.

10. An apparatus comprising: a comparator circuit to compare a memory request address with cache tags from set `n`, said comparator circuit to determine whether a cache entry in said set `n` matches said memory request address, said comparator circuit to output a memory address of any matched entries; and a mask circuit coupled to said comparator circuit, said mask circuit to mask an outputted memory address and to determine whether said outputted memory address can cause a thrash condition.

13. The apparatus of claim 10 further comprising an input to receive said cache tags from a cache memory.

14. The apparatus of claim 10 wherein said apparatus is a cache controller.

16. A system comprising: a processor coupled to a cache; a memory coupled to said cache; a cache controller coupled to said cache and said processor, said cache controller comprising: a comparator circuit to compare a memory request address with cache tags from set `n`, said comparator circuit to determine whether a cache entry in said set `n` matches said memory request address, said comparator circuit to output a memory address of any matched entries; and a mask circuit coupled to said comparator circuit, said mask circuit to mask an outputted memory address and to determine whether said outputted memory address can cause a thrash condition.

17. The system of claim 16 wherein said cache controller further comprises a latch to store a thrash bit mask, said latch coupled to said mask circuit and to supply said thrash bit mask to said mask circuit.

18. The system of claim 16 wherein said cache controller further comprises an input to receive said memory request address from said processor.

19. The system of claim 16 wherein said cache controller further comprises an input to receive said cache tags from said cache.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 3. Document ID: US 6519310 B2

L3: Entry 3 of 23 File: USPT Feb 11, 2003

DOCUMENT-IDENTIFIER: US 6519310 B2
TITLE: Hardware event based flow control of counters

Brief Summary Text (5):
An integrated circuit (IC) chip may use performance counters to monitor the performance of the chip in various ways. A performance counter may be used to count the number of data events occurring in a predetermined period of time at some regular or random interval. For example, some conventional processors have counters which count the number of instructions that have been run by the processor and/or the number of cache hits so that a ratio may be calculated of the number of cache bits per instruction.

Brief Summary Text (7):
As an example, in graphics controllers it can be valuable to know the number of certain events (i.e., polygons filled) on a per frame basis. But the hardware of conventional graphics controllers does not support event based sampling. Therefore, a software controlled method for facilitating graphics event controlled sampling is used wherein special instructions are injected into the graphics software driver to notify the graphics controller hardware when to take a sample. Because of the need for interaction with the graphics software driver, software based event based sampling is limited to a few graphic events and very limited in its capabilities.

Detailed Description Text (2):
An embodiment of the present invention seeks to provide hardware based event flow control of performance counters in an integrated circuit chip. An example application of the invention is in a processor or chipset of a desktop computer having performance counters for performance monitoring. In particular, an example embodiment of the invention can be incorporated in the silicon of a microprocessor or a component of a chipset, such as a Memory Controller Hub (MCH).

Detailed Description Text (3):
The example embodiment of the invention may of course be applied in any chip or chipset (with or without integrated graphics) and indeed wherever hardware event based sampling is desired for whatever reason. As an example, the number of cache hits that occur between the time a cache flush occurs and the first cache miss occurs could be counted. As yet another example, a performance counter could be used to facilitate hardware queue analysis. In particular, it is a feature of the invention that it can be implemented in a generic, scalable, "counter block" architecture that is flexible enough to be customized for inclusion in most integrated circuit chip designs without any changes in the core counter block for different implementations.

Detailed Description Text (5):
The example embodiment of the invention consists of a reusable module that minimizes hardware and software resource requirements when it is integrated as part of a component. The module is especially well suited for the chipsets of desktop computer, mobile computer, or servers, either with or without graphics integrated in the chipset. The module is flexible enough to scale down to a silicon overhead of approximately 5000-7,000 gates. Routing overhead is determined by which events are made to be observable in a particular implementation.

Detailed Description Text (24):
As one example application of the invention, the number of cache hits (event A) that occur between the time a cache flush (event B) occurs and the first cache miss (event C) occurs can be counted. The block diagram of this example is illustrated in FIG. 3 and the various waveforms are illustrated in FIGS. 4(a)-4(e). The desired counter blocks are programmed to start counting the cache hit events (event A shown in FIG. 4(b)) by incrementing by one for each cache hit event (303). For the counter 306 to actually begin counting and increment, the start command must first be triggered to begin by detecting an occurrence of a cache flush (event B shown in FIG. 4(c)). Immediately following that command the counter block controller is programmed to stop counting the events and latch the counted value (shown in FIG. 4(e)) into data register 305. However, execution of this command is conditional upon a cache miss occurring (event C in FIG. 4(d)). Thus, the cache miss command becomes the Command Trigger 302 that must occur before the number of cache hits would no longer be counted.

Detailed Description Text (28):
With hardware support for complete event based control as described in the example embodiments above, any event (graphics or otherwise) can control the counters without any interaction with a software driver even though in some cases it may be desirable to have the software checking the indicator bits as outlined in the previous example. This new functionality goes beyond the limitations in conventional systems to enable truly dynamic control of performance counters. Although an example embodiment is described above, the invention is not limited in its application to any particular processor or chipset. Indeed, an important aspect of the invention is that it is particularly useful for any silicon device employing large registers and counters for counting and measuring successive data values.

CLAIMS:

8. The integrated circuit chip recited in claim 7, wherein the associated counter counts the number of cache misses between a cache flush and a cache hit.

10. The integrated circuit chip recited in claim 1, wherein the integrated circuit chip comprises a memory controller hub.

18. The integrated circuit chip recited in claim 10, wherein the plurality of counters comprise a plurality of performance counters relating to the performance of said memory controller hub and said associated counter comprises an associated performance counter.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 4. Document ID: US 6510475 B1

L3: Entry 4 of 23 File: USPT Jan 21, 2003

DOCUMENT-IDENTIFIER: US 6510475 B1
TITLE: Data fetching control mechanism and method for fetching optimized data for bus devices behind host bridge

Brief Summary Text (7):
Accordingly, there is a need for an efficient data fetching control mechanism which fetches optimized data from a memory subsystem on one side of a host bridge such as PCI-PCI bridge for PCI devices on the other side of the host bridge in accordance with characteristics of a particular request, such as a command type, a data width, a clock frequency and a cache line size.

Detailed Description Text (3):
Attention now is directed to the drawings and particularly to FIG. 1, an example computer system platform having an example data fetching control mechanism incorporated therein according to the principles of the present invention is illustrated. As shown in FIG. 1, the computer system 100 may comprise a processor subsystem 110, a memory subsystem 120, connected to the processor subsystem 110 by a front side bus 10, graphics 130 connected to the memory subsystem 120 by a AGP or graphics bus 30, one or more host chipsets 140-150 connected to the memory subsystem 120 by hub links 40 and 50 for providing an interface with peripheral buses such as Peripheral Component Interconnect (PCI) buses 60 and 70 of different bandwidths and operating speeds, a flash memory 160, and a super I/O 170 connected to the chipset 150 by a low pin count (LPC) bus for providing an interface with a plurality of I/O devices 180, including, for example, a keyboard controller for controlling operations of an alphanumeric keyboard, a cursor control device such as a mouse, track ball, touch pad, joystick, etc., a mass storage device such as magnetic tapes, hard disk drives (HDD), and floppy disk drives (FDD), and serial and parallel ports to printers, scanners, and display devices. A different plurality of I/O devices 190 may be provided by the non-legacy PCI bus 60. In addition, it may be noted that the computer system 100 may be configured differently or employ some or different components than those shown in FIG. 1.

Detailed Description Text (4):

The processor subsystem 110 may include a plurality of host processors 110a-110n and a cache subsystem 112. The memory subsystem 120 may include a memory controller hub (MCH) 122 connected to the host processors 110a-110n by a front side bus 10 (i.e., host or processor bus) and at least one memory element 124 connected to the MCH 122 by a memory bus 20. The memory element 124 may preferably be a dynamic random-access-memory (DRAM), but may be substituted for read-only-memory (ROM), video random-access-memory (VRAM) and the like. The memory element 124 stores information and instructions for use by the host processors 110a-110n. The graphics 130 may be connected to the main controller hub 122 of the memory subsystem 120 by an AGP bus (graphics bus) 30, and may include, for example, a graphics controller, a local memory and a display device (e.g., cathode ray tube, liquid crystal display, flat panel display, etc.).

Detailed Description Text (9):
The data fetching control mechanism 146 may be integrated within the P64H 140 rather than provided as a separate chip within a host chipset for simplicity. Connected to the PCI bus 60 may be a group of PCI devices (I/O devices) 190, including, for example, multiple high performance peripherals in addition to graphics (motion video, LAN, SCSI, FDDI, hard disk drives, etc.). Also connected to the PCI bus 60 may be additional secondary bridges for providing data transfers between the PCI bus 60 and a secondary bus so that the data may be used by various component circuits joined to the secondary bus. The secondary bus may be an ISA bus, an EISA bus, or a similar bus, any of which typically transfers data at a rate slower than the PCI bus 60. Examples of secondary bridges may include those described in detail in a publication entitled "82420/82430 PCIset, ISA and EISA Bridges," in 1993, Intel Corporation.

Detailed Description Text (13):
Accordingly, the present invention advantageously provides an efficient and cost-effective data fetching control mechanism 146 implemented to fetch (or pre-fetch) optimized data from a main memory 124 of the memory subsystem 120 on one side (side A) of a host bridge such as a P64H 140 for PCI devices 190 on the other side of the P64H 140 in accordance with a particular request, such as a command type, a bus data width, a bus frequency and a cache line size. The PCI bus command type may be a 4-bit command (such as, for example: "0110", "1110" and "1100") which indicates either a "memory read" command, a "memory read line" command, and a "memory read multiple" command as specified by the "PCI Local Bus Specification, Revision 2.2." For example, the "memory read" command may be used to read or fetch a single Dword of 32-bit block of data. The "memory read line" command may be used to read or fetch more than a Dword of 32-bit block of data up to the next cache line (a complete cache line). The "memory read multiple" command may be used to read or fetch more than one cache line before disconnecting.

Detailed Description Text (14):
In addition to the command type, the bus frequency, the bus data width and cache line size may be utilized to determine variable fetch sizes in order to fetch optimized data from main memory 124 of the memory subsystem 120 for PCI devices 190 on one side of the P64H 140. The bus frequency may indicate either 33 MHz or 66 MHz, based upon M66EN (66 MHz_ENABLE) used as an input. The bus frequency may run at 33 to 66 MHz if M66EN is asserted, and 0 to 33 MHz if M66EN is de-asserted. In other words, the M66EN may be used to indicate to the current PCI device (bus master) 190 whether a PCI bus 60 is operating at either 33 MHz or 66 MHz. The bus data width may indicate either 32 bits or 64 bits, based upon REQ64# assertion by the PCI device 190. The REQ64# may correspond to a 64-bit request. In other words, the data transfer may be at 64 bits if REQ64# is asserted by the current PCI device 190, and may be 32 bits if REQ64# is de-asserted by the current PCI device 190. Lastly, the cache line size may be either 32 bytes or 64 bytes, based upon an internal register in a host bridge such as P64H 140. The length of a cache line may be defined by the Cacheline Size register in Configuration Space which is initialized by configuration software of the host bridge.

Detailed Description Text (15):
The data fetching control mechanism 146 according to the principles of the present invention may be integrated within the P64H 140 rather than having a separate chip formed as portion of the P64H 140. Such a data fetching control mechanism 146 may be implemented by a cloud of logic which receives input variables such as the command type (memory read, memory read line, and memory read multiple), REQ#64, M66EN , and optional cache line size, and an index table which generates corresponding index or fetch values indicating a fetch size for fetching data from main memory 124 of the memory subsystem 120 on one side (side A) of the P64H 140 for PCI devices 190 on the other side (side B) of the P64H 140, based upon the command type, the bus frequency, and the bus data width as will be described with reference to FIGS. 2 and 3, and alternatively, based upon the

command type, the bus frequency, the bus data width and cache line size as will be described with reference to FIGS. 2 and 4 hereinbelow.

Detailed Description Text (16):
Refer now to FIG. 2, a block diagram of an example data fetching control mechanism 146 for fetching optimized data for PCI devices 190 behind a host bridge such as a P64H 140 for a particular request according to the principles of the present invention is illustrated. As shown in FIG. 2, the data fetching control mechanism 146 may comprise a cloud (array) of logic 210 which receives input variables such as the command type (memory read, memory read line, and memory read multiple), REQ#64 and M66EN signals, and also optional cache line size, and an index table 220 which generates corresponding index or fetch values indicating a fetch size of data to be fetched from main memory 124 of the memory subsystem 120 on side A of the P64H 140 for a requesting PCI device 190 on side B of the P64H 140.

Detailed Description Text (17):
In a preferred embodiment of the present invention, the cloud of logic 210 which receives input variables of the command type (memory read, memory read line, and memory read multiple), REQ#64, M66EN (PCI frequency) and cache line size, and the index table 222 which generates output fetch values may be implemented by a programmable logic array (PLA) or a look-up table as shown in FIGS. 3 and 4. In particular, FIG. 3 illustrates one example look-up table implementation without reference to the cache line size of an example data fetching control mechanism shown in FIG. 2. Likewise, FIG. 4 illustrates another example look-up table implementation with reference to the cache line size of an example data fetching control mechanism shown in FIG. 2. The fetch values provided from the look-up table may be based upon latency calculations of the computer system platform. The fetch values provided may be specific to the computer system platform described with reference to FIG. 1. Accordingly, these fetch values provided may be varied with different computer system platforms, and may not be narrowly confined by those exemplary values described hereinbelow.

Detailed Description Text (18):
As shown in FIG. 3, the example look-up table 146 may be an array of 6 columns by 8 rows, and may include permutations of input variables of the bus frequency (33 MHz or 66 MHz), REQ64# (bus data width of either 32 bits when de-asserted or 64 bits when asserted), and the read command type (memory read, memory read line, and memory read multiple) in the first three columns, and corresponding fetch values for hard delayed transaction (DT) and soft delayed transaction (DT) requests in the last three columns. In this example embodiment, the Cacheline Size register is not implemented. As a result, the PCI device 190 on the PCI bus 60 may assume that a cache line size of 32 bytes and use the read commands as described. A hard delayed transaction (DT) is an initial transaction when a read request is received from a PCI device 190, and starts when a read cycle is absorbed by the P64H 140. The request data length indicates the number of 32-bytes lines that the P64H 140 may request on the primary PCI bus (i.e., hublink 40) as a result of the input variables such as the bus frequency, REQ64# and the command type. A soft delayed transaction (DT) is an algorithm which pre-fetches additional lines of data in case where additional data is requested. The lines remaining indicates the number of lines must be left before additional lines may be requested from the P64H 140.

Detailed Description Text (21):
FIG. 4 illustrates an alternative example look-up table implementation of an example data fetching control mechanism 146 as shown in FIG. 2. As shown in FIG. 4, the example look-up table 146 may include permutations of input variables of the cache line size of 64 bytes, bus frequency (33 MHz or 66 MHz), REQ64# (bus data width of either 32 bits when de-asserted or 64 bits when asserted), the read command type (memory, memory read line, and memory read multiple) in the first four columns, and corresponding fetch values for hard delayed transaction (DT) and soft delayed transaction (DT) requests in the last three columns. In this alternative embodiment, the Cacheline Size register is implemented. As a result, the cache line size is a 64 bytes line. All odd data line outputs for hard DT and soft DT requests may turn into next even line numbers in order to avoid multiple snoop phenomenon. This is important since odd data line requests may cause multiple snoops during the same read cycle.

Detailed Description Text (26):
Upon reset of the computer system 100, state machine 500 assumes an initial state labeled FREE state 502. The state machine 500 remains idle and is free to receive new delayed transaction (DT) cycle. During FREE state 502, state machine 500 monitors the loading of a hard delayed transaction (DT) and moves to HWAIT state 504, when a hard DT

has been loaded. During HWAIT state 504, state machine 500 launches a hard DT request on the primary PCI bus as a result of input variables as shown, for example, in FIGS. 3 and 4, in order to obtain the output request data length for hard DT shown, for example, in first output column of FIGS. 3 and 4. After the output request data length for hard DT is obtained, state machine 500 transitions from HWAIT state 504 to MEMRUN state 506 to allow a requesting PCI device 190 (PCI master) to retrieve data fetched from the memory subsystem 120. When the number of cache lines has fallen below the lines remaining watermark shown in second output column of FIGS. 3 and 4, state machine 500 transitions from MEMRUN state 506 to SREG state 508 to issue a soft DT request in order to obtain the output request data length for soft DT shown, for example, in last output column of FIGS. 3 and 4.

Detailed Description Text (27):
After the output request data length for soft DT is obtained, state machine 500 transitions from SREG state 508 to SRUN state 510 to allow the requesting PCI device 190 (PCI master) to retrieve data fetched from the memory subsystem 120. The state machine 500 may transition from SRUN state 510 back to SREG state 508 any time the number of cache lines has fallen below the lines remaining watermark shown in second output column of FIGS. 3 and 4. However, whether the state machine 500 is in either MEMRUN state 506, SREG state 508 or SRUN state 510, hard or soft delayed transaction (DT) may be terminated because certain invalidation conditions have occurred. For example, state machine 500 may transition from MEMRUN state 506 to INVALID state 512 if hard DT entry has become invalid for one of the following reasons: (1) the requesting PCI device 190 (PCI master) has terminated the DT; (2) hard DT timer has expired; or (3) all DT data has been retrieved which may prevent issuance of soft DT requests. Similarly, state machine 500 may transition from either SREG state 508 or SRUN state 510 to INVALID state 512 if soft DT entry has become invalid for one of the following reasons: (1) the requesting PCI device 190 (PCI master) has terminated the DT; (2) soft DT timer has expired; (3) the snoop hit from the requesting PCI device 190 (PCI master) writes to the same 4KB page boundary; (4) there was a downstream request before the request was able to be issued upstream; or (5) all DT data has been retrieved which may prevent issuance of soft DT requests.

Detailed Description Text (31):
While there have been illustrated and described what are considered to be exemplary embodiments of the present invention, it will be understood by those skilled in the art and as technology develops that various changes and modifications may be made, and equivalents may be substituted for elements thereof without departing from the true scope of the present invention. For example, the computer system as shown in FIG. 1 may be configured differently or employ some or different components than those illustrated. In addition, the data fetching control mechanism shown in FIGS. 2-4 may be configured differently or employ some or different components than those illustrated without changing the basic function of the invention. For instance, different combinations of logic gates may be used to correspond input variables such as the command type (memory read, memory read line, and memory read multiple), REQ#64 and M66EN signals, and also optional cache line size with output fetch values for fetching data from a memory subsystem on one side of a host bridge to a designated PCI device on the other side of the host bridge. Additionally, alternative bus widths and frequencies may be used as both bus widths and bus frequencies tend to increase as technology advances. Further, software equivalents to the data fetch control mechanism as shown in FIGS. 2-4 may be available to fetch data from a memory subsystem on one side of a host bridge to the PCI devices on the other side of the host bridge. Many modifications may be made to adapt the teachings of the present invention to a particular situation without departing from the scope thereof Therefore, it is intended that the present invention not be limited to the various exemplary embodiments disclosed, but that the present invention includes all embodiments falling within the scope of the appended claims.

CLAIMS:

7. The mechanism as claimed in claim 2, wherein said input logic is further coupled to receive information pertaining a cache line size of a computer system for said control logic to generate the fetch values indicating the fetch sizes of data to be fetched from said memory subsystem.

15. The computer system as claimed in claim 8, wherein said data fetching control mechanism further receives information pertaining a cache line size of the computer system for generating the fetch values indicating the fetch sizes of data to be fetched from said memory subsystem.

22. The method as claimed in claim 17, further comprising: receiving information pertaining a cache line size of the computer system; and generating said fetch values indicating the fetch sizes of data to be fetched from said memory subsystem in correspondence with the input varables of said read command, said bus frequency, said bus data width and said cache fine size from said bus device.

25. The computer system as claimed in claim 23, wherein the look-up table further receives information pertaining a cache line size of the computer system and produces the fetch values indicating the fetch sizes of data to be fetched from the memory in accordance with the input variables of the read command, the bus frequency, the bus data width and the cache line size from the bus device.

30. The computer system as claimed in claim 29, wherein the look-up table further receives information pertaining a cache line size of the computer system and produces the fetch values indicating the fetch sizes of data to be fetched from the memory in accordance with the input variables of the read command, the bus frequency, the bus data width and the cache line size from the bus device.

☐ 5. Document ID: US 6507904 B1

L3: Entry 5 of 23 File: USPT Jan 14, 2003

DOCUMENT-IDENTIFIER: US 6507904 B1
TITLE: Executing isolated mode instructions in a secure system running in privilege rings

Detailed Description Text (4):
The parameter storage is one of an internal storage and an external storage. The internal storage includes a control register to store the execution mode word, a logical processor register, a mask register to store the mask value, a base register to store the base value, a frame register set, an exit frame register to store the exit address, an entry frame register to store the entry address, and a loader register to store the processor nub loader address. The external storage includes a memory controller hub (MCH) storage and an input/output controller hub (ICH) storage. The execution mode word configures the processor in the isolated execution mode.

Detailed Description Text (14):
One concept of the isolated execution architecture is the creation of an isolated region in the system memory, referred to as an isolated area, which is protected by both the processor and chipset in the computer system. The isolated region may also be in cache memory, protected by a translation look aside (TLB) access check. Access to this isolated region is permitted only from a front side bus (FSB) of the processor, using special bus (e.g., memory read and write) cycles, referred to as isolated read and write cycles. The special bus cycles are also used for snooping. The isolated read and write cycles are issued by the processor executing in an isolated execution mode. The isolated execution mode is initialized using a privileged instruction in the processor, combined with the processor nub loader 52. The processor nub loader 52 verifies and loads a ring-0 nub software module (e.g., processor nub 18) into the isolated area. The processor nub 18 provides hardware-related services for the isolated execution.

Detailed Description Text (21):
FIG. 1C is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 includes a processor 110, a host bus 120, a memory controller hub (MCH) 130, a system memory 140, an input/output controller hub (ICH) 150, a non-volatile memory, or system flash, 160, a mass storage device 170, input/output devices 175, a token bus 180, a motherboard (MB) token 182, a reader 184, and a token 186. The MCH 130 may be integrated into a chipset that integrates multiple functionalities such as the isolated execution mode, host-to-peripheral bus interface, memory control. Similarly, the ICH 150 may also be

integrated into a chipset together or separate from the MCH 130 to perform I/O functions. For clarity, not all the peripheral buses are shown. It is contemplated that the system 100 may also include peripheral buses such as Peripheral Component Interconnect (PCI), accelerated graphics port (AGP), Industry Standard Architecture (ISA) bus, and Universal Serial Bus (USB), etc.

Detailed Description Text (25):
The host bus 120 provides interface signals to allow the processor 110 or processors 110, 100a, and 110b to communicate with other processors or devices, e.g., the MCH 130. In addition to normal mode, the host bus 120 provides an isolated access bus mode with corresponding interface signals for memory read and write cycles when the processor 110 is configured in the isolated execution mode. The isolated access bus mode is asserted on memory accesses initiated while the processor 110 is in the isolated execution mode. The isolated access bus mode is also asserted on instruction pre-fetch and cache write-back cycles if the address is within the isolated area address range and the processor 110 is initialized in the isolated execution mode. The processor 110 responds to snoop cycles to a cached address within the isolated area address range if the isolated access bus cycle is asserted and the processor 110 is initialized into the isolated execution mode.

Detailed Description Text (32):
I/O devices 175 may include any I/O devices to perform I/O functions. Examples of I/O devices 175 include a controller for input devices (e.g., keyboard, mouse, trackball, pointing device), media card (e.g., audio, video, graphics), a network card, and any other peripheral controllers.

CLAIMS:

5. The apparatus of claim 4 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

19. The method of claim 18 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

33. The system of claim 32 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

47. The computer program product of claim 46 wherein the external storage includes a memory controller hub (MCH) storage in an MCH and an input/output controller hub (ICH) storage in an ICH.

☐ 6. Document ID: US 6505259 B1

L3: Entry 6 of 23 File: USPT Jan 7, 2003

DOCUMENT-IDENTIFIER: US 6505259 B1
TITLE: Reordering of burst data transfers across a host bridge

Brief Summary Text (5):
Typical peripheral standardized buses include the Peripheral Component Interconnect (PCI) bus or bridge that links devices or agents such as video devices, disk drives, and other adapter cards. A second bus often used in connection with a PCI bus in modern computer systems is the Accelerated Graphics Port (AGP). AGP is an interface specification generally designed for the throughput demands of 3-D graphics.

Brief Summary Text (6):

Communication protocol between a processor and peripheral devices or agents about a peripheral bus generally allows the transfer of chunks of data of 8 bytes or less. Such chunks represent a quad word. In addition to quad words, communication protocols also allow the transfer of data as four quad words or 32 bytes. Such a transfer is referred to as a cache line or burst. A cache line or burst transfer is typically faster than a transfer of four individual quad words of the same data, because the transfer of a burst allows compacting of the data.

Brief Summary Text (7):
When a processor reads memory, the processor requests a section of address space in memory. That address space may typically be represented by a quad word. Typically, what the processor receives in response to its request is a cache line or burst that includes the requested quad word. The burst order refers to the choice of addresses for the sequence of a burst or cache line. In modern systems, the receipt of a burst does not necessarily correspond to the sequentially ordered quad words that make up the burst in memory space. Instead, the line is returned with the requested quad word first, followed by the remaining quad words toggled in a non-linear fashion as known in the art.

Brief Summary Text (8):
The above description related to a processor requesting data from memory over, for example, a memory bus. The same communication protocol is followed when a processor requests data from a peripheral device or agent. Data returned to a processor as part of a read transaction initiated by the processor is returned as a burst or cache line that may or may not represent a sequential transfer of data from a cache line or burst. One problem is systems that utilize a PCI bus as a communication link between the peripheral device or agent and the processor is that PCI generally only understands sequential or linear ordering. Thus, a non-sequential or non-linear burst transaction initiated by a processor is returned to the processor as four distinct requests for data (four quad words). Thus, the efficiency of the system is limited by PCI's inability to transfer continuous bursts of data in non-linear order.

Detailed Description Text (3):
FIG. 1 illustrates a computer system incorporating the transfer method of an embodiment of the invention in general block diagram form. Computer system 10 includes processor 100 (and optionally processor 110 and other processors) coupled to memory controller hub (MCH) 120. In one aspect, MCH 120 controls the accessing of memory 130 over memory bus 140. In this example, also coupled to MCH 120 is accelerated graphics port (AGP) 160 to communicate chiefly with advanced video and other graphic devices 150 and 155.

Detailed Description Text (5):
In the following example, a processor read transaction will be described. In one example, a line transfer reads or writes a cache line or burst. On a processor such as a Pentium.RTM. Pro processor, commercially available from Intel Corporation of Santa Clara, Calif., a cache line or burst is 32 bytes aligned on a 32-byte boundary. As noted above, while a line is always aligned on a 32-byte boundary, a line transfer need not begin on that boundary. A cache line or burst is transferred in four 8-byte chunks or quad words, each of which can be identified by a certain address bit. Table 1 illustrates an exemplary transfer order used for a 32-byte line, based on address bits A[4:3]# specified in a transaction Request Phase.

Detailed Description Text (15):
By reordering a transaction, the invention offers improved performance of transactions over buses that are not suited for non-linear cache line contiguous transfers. The invention allows requested non-linear cache lines or bursts to be transferred across secondary buses as linear cache lines or bursts thus reducing the design complexity of prior art systems that break the burst into smaller chunks of data.

CLAIMS:

1. A method comprising: reordering a non-linear burst transaction initiated by a processor targeting a peripheral bus to a linear order; retrieving the linear burst from the peripheral bus; and returning the initiated non-linear burst to the processor; wherein a memory controller hub (MCH) reorders the non-linear burst initiated by the processor and retrieves the linear burst transaction before returning the transaction to the processor as a non-linear burst.

4. The method of claim 1, wherein the bus is selected from the group consisting of an Accelerated Graphics Port (AGP) and a Peripheral Component Interconnect (PCI) bus.

7. The apparatus of claim 6, wherein the bus bridge comprises a memory controller hub (MCH).

8. The apparatus of claim 5, wherein the bus is selected from the group consisting of an Accelerated Graphics Port (AGP) and a Peripheral Component Interconnect (PCI) bus.

13. The apparatus of claim 10, wherein the receiver is selected from the group consisting of an Accelerated Graphics Port (AGP) and a Peripheral Component Interconnect (PCI) bus.

14. A machine readable storage media containing executable computer program instructions which when executed cause a digital processing system to perform a method comprising: reordering a non-linear burst transaction initiated by a processor targeting a bus to a linear burst order; retrieving the linear burst order from the bus; and returning the linear burst order to the processor as the initiated non-linear burst; wherein a memory controller hub (MCH) re-orders the non-linear burst initiated by the processor and retrieves the linear burst order transaction before returning the transaction to the processor as a non-linear burst.

☐ 7. Document ID: US 6496888 B1

L3: Entry 7 of 23 File: USPT Dec 17, 2002

DOCUMENT-IDENTIFIER: US 6496888 B1
TITLE: Incorporation of bus ratio strap options in chipset logic

Detailed Description Text (6):
Referring now to FIG. 2, there is a block diagram of a computer system 200 that includes the incorporation of bus ratio strap options in chipset logic. Sample system 200 may incorporate bus ratio strap options in chipset logic, in accordance with the present invention, such as in the embodiment described herein. In alternative embodiments, system 200 may comprise of applications such as system on a chip, digital signal processing system, or cellular phones. Sample system 200 is representative of processing systems based on the PENTIUM.RTM., PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 200 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (8):
System 200 includes a memory 216. Memory 216 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, or other memory device. Memory 216 may store instructions and/or data represented by data signals that may be executed by processor 202. A cache memory 204 may reside inside processor 202 that stores data signals stored in memory 216. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (9):
In this embodiment, chipset logic 214 is coupled to the processor bus 210 and memory 216. Bus ratio strap options 218 resides in chipset logic 214. Chipset logic 214 in system 200 provides the function of a bridge/memory controller. Chipset logic 214 directs data signals between processor 202, memory 216, and other components in the system 200 and bridges the data signals between processor bus 210, memory 216, and a first input/output (I/O) bus 220. In some embodiments, the bridge/memory controller provides a graphics port for coupling to a graphics controller 212. In one embodiment, the bridge/memory controller 214 may be referred to as the North Bridge portion of the chipset logic.

Detailed Description Text (20):
During normal operating state, the processor performs usual tasks such as executing program instructions, performing calculations, and processing data. However, the user may decide to change the system to a higher performance state while the system is up and running in a normal operating mode. FIG. 4b is a flow diagram showing a bus ratio change sequence. The processor first writes out commands to the chipset logic to request a higher performance mode at step 450. The system is then placed in a suspend/sleep state at step 460. In one embodiment, the sleep state will cause the hardware to perform the following steps: stop system clocks, place system memory into a self or suspend refresh state, power off the processor and cache subsystem. When the system is in a sleep state, the memory cache may be flushed and the processor/system cache context lost. However, the context of the core logic and memory is maintained in one embodiment. When the processor is in a deep power-down or sleep mode, the processor operating clock frequency and clock ratio information may be changed. The processor returns to normal operating state from deep power-down upon detection of a wakeup event such as receiving any one of a number of interrupt or reset signals. For example, commonly known signals such as INTR, NMI, SMI#, RESET, or INIT may cause the processor to transition back to a normal operating state. As the system comes out of suspend or sleep, the chipset logic recognizes the system resume and resets the processor at step 470. The chipset also recognizes that the processor has requested that a higher bus ratio be used. While the processor is in its reset sequence, the bus ratio register in the chipset drives the new bus ratio setting to the processor at step 480. As the processor returns to a normal operating state, the processor should be functioning at a higher performance state. Hence the system may alternate between high performance and low performance modes during operation.

CLAIMS:

15. The apparatus of claim 11 wherein said chipset comprises of a memory controller hub.

24. The digital processing system of claim 19 wherein said chipset comprises a memory controller hub.

☐ 8. Document ID: US 6496193 B1

L3: Entry 8 of 23 File: USPT Dec 17, 2002

DOCUMENT-IDENTIFIER: US 6496193 B1
TITLE: Method and apparatus for fast loading of texture data into a tiled memory

Brief Summary Text (2):
The present invention relates generally to the field of graphics processing; more particularly, to methods and apparatus directed to efficiently loading and accessing data in a graphics processing system.

Brief Summary Text (4):
A requirement of a modern computer graphics processing system is that it be capable of manipulating and performing calculations on huge quantities of data at very fast speeds. This has led researchers to search for better techniques for handling and storing data to be output to a display device in real-time.

Brief Summary Text (5):
The demand for more efficient methods and circuits for dealing with graphics data continues to grow rapidly. The increased use of techniques for three-dimensional texturing (solid texturing), modeling, and shading in computer graphics and animation has fueled this demand. For example, it is now possible for three-dimensional objects and shapes created on a personal computer or workstation to be filled in with textures

simulating natural materials and natural phenomena. Realistic images having textures resembling marble, wood, or stone are now commonly displayed. Likewise, computer researchers and scientists have successfully simulated natural phenomena such as fog, waves, fire, and clouds for display on a computer screen.

Brief Summary Text (6):
A "texture"--sometimes referred to as a "pattern" or "wallpaper"--is simply an image that is used as surface covering for objects or shapes created by a graphics program running on a computer. Textures come in many varieties. Examples include bump mapping, which simulate the appearance of surface bumps without actually modifying the geometry of the object; light maps which give the appearance of smooth diffuse lighting; displacement mapping, in which textures are used actually to move the surface; and solid textures that give consistent textures of all surfaces of an object regardless of distortions of the surface parameter space.

Brief Summary Text (7):
A texture may be explicitly specified by storing all the complex details of the scene or sequence in a bit-map that contains the value of each pixel (called a "texel` when referring to texture data) in the image. This obviously requires a large amount of storage space within the computer as well as an advanced graphics processing subsystem. Typically, the processor of a computer graphics subsystem loads the texture data into memory before is used by the graphics application. Since the end-user must wait for the texture data to be completely loaded into memory before starting the application (e.g., playing a video game), it is desirable to reduce this time as much as possible.

Brief Summary Text (8):
One architectural technique for increasing the memory access speed of graphics data is a "tiled" memory organization. In a tiled memory, the memory mapped for the display screen is translated into tiles within the memory (e.g., DRAM) in a way that permits fast access of neighboring pixels in the array. By way of example, U.S. Pat. No. 5,736,988 teaches an apparatus and method for storing graphics data as a collection of individual addressable tiles for fast retrieval in a computer system. As further background, U.S. Pat. Nos. 5,877,780 and 5,781,200 both disclose a tiled memory architecture for storing graphics data. The memory mapping taught involves a translation between a received linear address (representing a screen pixel location and color) to a physical address stored in within the DRAM.

Brief Summary Text (9):
U.S. Pat. No. 5,828,382 of Wilde describes a graphics subsystem that includes graphics hardware for permitting graphics texture maps to be dynamically cached internally within the graphics hardware. Textures are stored in system memory in a tile format that allows an entire cache tile to be stored linearly in memory space. According to Wilde, storing the textures in a tiled linear format allows the graphics processor to fetch the tile across the peripheral component interface (PCI) bus in a single burst cycle.

Brief Summary Text (11):
Despite the widespread use of tiled memories and improved processor performance, there is still a need for a solution to the problem of how to achieve fast loading of texture data in a computer graphics system.

Drawing Description Text (3):
FIG. 1 is a diagram of one possible implementation of a computer graphics processing system in accordance with the present invention.

Detailed Description Text (3):
In a personal computer graphics subsystem, texture mapping is an affordable technique to bring richness and reality into graphical display scenes. Typically, the central processing unit (CPU) loads the textures into the memory before the application program begins using the data. For the end-user, it is critical to reduce the loading time, as he must wait for the textures to be loaded before the application can start. The present invention solves this problem by providing fast texture loading within an efficient hardware implementation.

Detailed Description Text (4):
Referring to FIG. 1, a general block diagram of one embodiment of a graphics processing system 10 implemented in accordance with the present invention is shown. Graphics processing system 10 includes a processor 11 coupled to a graphics memory controller hub 12. Essentially, graphics memory controller 12 functions as a first bridge element

providing a communication path between host processor 110, system memory 14, graphics memory 13, and various display devices, which may include a video graphics accelerator cathode ray tube (VGA CRT) device 16 and/or a digital display 15. Graphics memory 13 is coupled to graphics memory controller 12 via a dedicated graphics bus 24, whereas system memory 14 is coupled to graphics memory controller 12 of via system bus 25. Graphics memory 13 and system memory 14 are both arranged as tiled memories in accordance with one embodiment of the present invention.

Detailed Description Text (6):
Graphics memory controller hub 12 also interfaces with an input/output (I/O) controller hub 17 in the system diagram of FIG. 1. An assortment of peripheral devices may be coupled to I/O controller 17 via a Peripheral Component Interconnect (PCI) bus. These peripherals are represented in the example of FIG. 1 as multiple PCI agents 22 and devices connected to a plurality of PCI slots 23. These peripheral devices may include one or more graphics controllers or a network interface card coupled to a private internal network.

Detailed Description Text (8):
Additionally, a non-volatile memory element 29 may be coupled to I/O controller 17. Non-volatile memory element 29, which may comprise a flash memory, is ordinarily utilized to store basic input/output system (BIOS) information used by the graphics system 10 during its boot sequence.

Detailed Description Text (10):
As discussed previously, a tiled memory architecture is an efficient method of utilizing memory space in a memory subsystem. Tiling allows the graphics engine to access memory faster without causing an excessive number of page misses in the memory subsystem. Graphics virtual memory can be tiled into different ways: X-major tiling and Y-major tiling. In X-major tiling, two contiguous quadwords in the X-direction are consecutive in physical memory. (Note that a quadword denotes four words, with each word comprising two bytes of data. An octalword denotes eight words, or 16 bytes of data.)

Detailed Description Text (13):
Practitioners familiar with the computer graphics arts will appreciate that the type of tiling used for any given surface depends on the operand access pattern exhibited by the related graphics engine. That is, the tiling structure is established and known by the graphics driver hardware. The host processor lacks understanding of the tiling structure generated in the graphics memory. Basically, what this means is that when the processor receives an instruction to load texture data into a particular area of memory, it simply fetches the tiling address and begins sending out the texture data. However, since the memory is tiled, there is a need for an operation to convert the virtual addresses that the host processor is writing to the tiled address space, and thus the physical address space. It is this need that is satisfied by the present invention.

Detailed Description Text (14):
The texture engine, which typically resides within the graphics processor, usually operates on a square quantity of texels. Based on the depth of the texel, it normally accesses two contiguous quadwords in the Y-direction. The present invention advantageously recognizes that Y-major tiling is more efficient for the texture engine, and therefore loads texture data from the processor into memory in the Y-direction.

Detailed Description Text (15):
It is important to recognize that when the texture is loaded into a X-major tiled surface a single cache line (i.e., four quadwords) at a time, four quadwords are contiguous in memory. This means that a single request can be issued to the memory controller to load the cache line of data. On the other hand, when the texture is a Y-major tiled surface that is not the case; the requests for the four quadwords need to be fragmented before tiling the address. Fragmentation, according to the present invention, is the act of generating individual addresses for each quadword, given that the input data stream consists of a four-quadword chunk having a specified starting address. The present invention first fragments in the desired order, and then tiles the addresses before they are converted to physical addresses (using a translation look-aside buffer).

Detailed Description Text (16):
In the presently described embodiment, the host processor generates addresses for writing a cache line of texture data in accordance with four different sequences. These

sequences are listed in the table shown in FIG. 3. The first sequence, sequence "A", starts with quadword 0, followed by quadwords 1, 2 and 3. Sequence "B" has a quadword ordering of 1, 0, 3 and 2. Sequence "C" has a quadword ordering of 2, 3, 0 and 1. And sequence "D" has a quadword ordering of 3, 2, 1 and 0. These four different sequences denote the four different quadword orderings obtained from the cache line of data, corresponding to the different ways that the processor outputs the texture data.

Detailed Description Text (17):
A finite state machine implementation of the present invention is shown in FIG. 4. States 0-3 represent the starting point corresponding to the four different sequences described above. That is, based on the starting address, the finite state machine begins in one of these four states; it then generates the output addresses for the quadwords as it transitions around to each of the different states. For example, if the starting address indicated quadword sequence "A", the entry point for the finite state machine would be state 0. If the starting address indicated quadword sequence "B", the entry point for the finite state machine would be state 1, and so on. Sequencing in this manner is beneficial in the storage of texture data into a tiled memory since it can be accessed quickly by the graphics processor, and also written to quickly by the host processor.

Detailed Description Text (20):
Thus, a two flip-flop logic embodiment covers all of the four orderings of cache line data listed in the table of FIG. 3. This simple, yet elegant implementation makes processor address ordering for the loading of texture data on a par with regular addressing in terms of speed and resources.

CLAIMS:

8. A graphics processing system for fast loading of texture data comprising: a host processor; a graphics memory controller coupled to the host processor; a tiled memory coupled to the graphics memory controller; wherein the host processor includes state machine logic to generate a sequence of addresses for writing a cacheline of texture data into the tiled memory according to Y-major tiling, the cacheline comprising quadwords (QWs) 0-3, the sequence corresponding to an ordering of the QWs 0-3 consisting of either: (a) QW0, QW1, QW2, QW3; (b) QW1, QW0, QW3, QW2; (c) QW2, QW3, QW0, QW1; or (d) QW3, QW2, QW1, QW0, depending upon a starting address.

9. The graphics processing system of claim 8 further comprising: a display device coupled to the graphics memory controller.

10. The graphics processing system of claim 8 wherein the tiled memory comprises a graphics memory of DRAM devices.

11. The graphics processing system of claim 8 wherein the state machine logic comprises a pair of flip-flops.

12. The graphics processing system of claim 8 wherein the quadwords are contiguous.

13. The graphics processing system of claim 8 wherein the starting address comprises bits [31:0].

14. The graphics processing system of claim 13 wherein a first address in the sequence corresponds to the starting address and a second address in the sequence is generated from the first address by flipping bit 3 of the starting address.

15. The graphics processing system of claim 14 wherein a third address in the sequence is generated from the second address by flipping bit 3 and bit 4 of the second address.

16. The graphics processing system of claim 15 wherein a fourth address in the sequence is generated from the third address by flipping bit 3 of the third address.

☐ 9. Document ID: US 6496055 B2

L3: Entry 9 of 23 File: USPT Dec 17, 2002

DOCUMENT-IDENTIFIER: US 6496055 B2
TITLE: Gate enhanced tri-channel positive charge pump

Detailed Description Text (6):
Referring now to FIG. 1, an exemplary computer system 100 is shown. System 100 includes a component, such as a processor, employing a gate enhanced tri-channel positive charge pump in accordance with the present invention, such as in the embodiment described herein. System 100 is representative of processing systems based on the PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 100 may be executing a version of the WINDOW.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (9):
System 100 includes a memory 120. Memory 120 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102. A cache memory 104 can reside inside processor 102 that stores data signals stored in memory 120. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (10):
A system logic chip 116 is coupled to the processor bus 110 and memory 120. The system logic chip 116 in the illustrated embodiment is a memory controller hub (MCH). The processor 102 communicates to the MCH 116 via a processor bus 110. The MCH 116 provides a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 116 directs data signals between processor 102, memory 120, and other components in the system 100 and bridges the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 provides a graphics port for coupling to a graphics controller 112. The MCH 116 is coupled to memory 120 through a memory interface 118. The graphics card 112 is coupled to the MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

Detailed Description Text (12):
A tri-channel triple well positive charge pump 126 also resides in flash memory BIOS 128. Alternate embodiments of a gate enhanced tri-channel charge pump 126 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits.

Detailed Description Text (13):
For another embodiment of a system, one implementation of a charge pump can be used with a system on a chip. One embodiment of a system on a chip comprises of a processor and a memory. The memory for one system is a flash memory. The flash memory can be located on the same die as the processor and other system components. Additionally, other logic blocks such as a memory controller or graphics controller can also be located on a system on a chip. By including one embodiment of the present invention on the system on a chip, the flash memory can be enabled to program and erase flash memory cells without requiring a high voltage pin on the system on a chip pin-out. The needed high voltage potentials can be generated on the same die.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

☐ 10. Document ID: US 6480200 B1

L3: Entry 10 of 23 File: USPT Nov 12, 2002

DOCUMENT-IDENTIFIER: US 6480200 B1
TITLE: Method and apparatus for deferred texture validation on a multi-tasking computer

Abstract Text (1):
A method of avoiding processor state transitions while rendering frames includes forming a command data buffer and a data structure in response to drawing commands received from a graphics application. The command data buffer includes a list of primitives. The data structure identifies a texture and where in the command data buffer (offset) the texture should be referenced. Multiple texture changes can be accumulated at user level before making a ring transition to kernel mode to process the command data buffer. After the command data buffer and data structure are formed, kernel mode is entered so that a graphics driver can communicate the command data buffer to graphics hardware. While in kernel mode offsets stored in the data structure cause the driver to verify that the texture is present in graphics memory. If not present, the driver causes the texture to be loaded before continuing to communicate the primitives. Thus, multiple textures can be loaded the kernel session.

Brief Summary Text (5):
Computer graphics systems are commonly used for displaying two-and three-dimensional graphics representations of objects on a two-dimensional video display screen. Current computer graphics systems provide highly detailed representations and are used in a variety of applications, such as computer-aided design, animation and the like.

Brief Summary Text (6):
In a typical computer graphics system, an object or model to be represented on the display screen is broken down into graphics primitives. Primitives are basic components of a graphics display and may include, for example, points, lines, triangles, quadrilaterals, triangle strips and polygons. Typically, a hardware/software scheme is implemented to render, or draw, the graphics primitives that represent a view of one or more objects being represented on the display screen.

Brief Summary Text (8):
Early graphics systems failed to display images in a sufficiently realistic manner to represent or model complex three-dimensional objects. The images displayed by such systems exhibited extremely smooth surfaces absent textures, bumps, scratches, shadows and other surface details present in the object being modeled.

Brief Summary Text (9):
As a result, methods were developed to display images with improved surface detail. Texture mapping is one such method to improve surface detail that involves mapping a source image, referred to as a texture, onto a surface of a three-dimensional object, and thereafter mapping the textured three-dimensional object to the two-dimensional graphics display screen to display the resulting image. Surface detail attributes commonly texture mapped include color, specular reflection, vector perturbation, specularity, transparency, shadows, surface irregularities and grading. Texture mapping involves applying one or more point elements (texels) of a texture to each point element (pixel) of the displayed portion of the object to which the texture is being mapped. Texture mapping hardware subsystems typically include a local memory cache that stores texture mapping data associated with the portion of the object being rendered.

Brief Summary Text (10):
The basic components of a computer graphics system typically include a host computer and graphics hardware. The host computer may include a graphics application, a graphics application programming interface and a graphics driver. The graphics application programming interface (API) receives commands from a user through the graphics application program and provides primitive data, in conjunction with the graphics driver, to the graphics hardware. The graphics hardware typically includes a geometry accelerator, a rasterizer and a frame buffer, and may include texture mapping hardware. The geometry accelerator receives primitive data from the host computer and performs operations such as coordinate transformations and lighting, clipping and plane equation calculations for each primitive. The output of the geometry accelerator, referred to as rendering data, is used by the rasterizer and the texture mapping hardware to generate final screen coordinate and color data for each pixel in each primitive. The pixel data from the rasterizer and the pixel data from the texture mapping hardware are combined

and stored in the frame buffer for display on a display output device such as a monitor, TV or flat panel display.

Brief Summary Text (11):
The applications in the host computer frequently utilizes the OpenGL application programming interface standard. (OpenGL is a registered trademark of Silicon Graphics, Inc.). The OpenGL standard is a hardware independent interface designed to be implemented on different hardware platforms that provides a complete library of low level graphics manipulation commands for describing models of three-dimensional objects. This standard was originally based on the proprietary standards of Silicon Graphics, Inc., but was later transformed into an open standard, which is used in high end graphics-intensive workstations and, more recently, in high end personal computers. OpenGL is supported on a wide range of hardware platforms and software environments such as, Mac.RTM. OS, OS/2.RTM., UNIX, Solaris, Windows 95/98, Windows NT, Linux, OPENStep, Python, and BeOS. The OpenGL standard is described in the OpenGL Programming Guide, Version 1.2 (1999) and the OpenGL Reference Manual, Version 1.1 (1997).

Brief Summary Text (12):
Applications that use texture mapping typically communicate texture map data prior to the start of rendering an image. This is done through glTexImage commands that are issued to the API. To render the image OpenGL commands are passed to the API that specify primitive data and what texture data, if any, is associated with the primitives. Texture map data is associated with a primitive or primitive using the glBindTexture command. A glBindTexture command specifies which texture will be applied to all subsequent primitives until a new texture is specified or texturing is disabled. Typically, the graphics driver processes the OpenGL commands to form a command/data buffer of low level commands according to the graphics hardware implementation. The graphics hardware reads the command/data buffer to render the image.

Brief Summary Text (13):
Before a primitive is rendered, the texture map must be resident on the graphics hardware. Because texture memory is finite, it is possible for the total amount of texture space needed by a graphics application (or multiple applications running simultaneously) to exceed the capacity of the texture memory. This may cause the graphics driver to load and unload textures to ensure that the desired texture is available in texture memory when needed. Thus, if the texture map is not resident in the graphics hardware when the command/data buffer is constructed, the graphics driver will cause the texture map to be loaded into the hardware. This is done by placing the texture data in a separate command/data buffer. The command/data buffer is loaded into the hardware by programming a direct memory access (DMA) engine with the address of the command/data buffer, and the destination address in the graphics hardware.

Brief Summary Text (14):
In Windows NT, all hardware device accesses must occur in privileged mode (or kernel mode). Since the graphics application is running in user mode, a ring transition from user mode to kernel mode is required every time a texture map is loaded. Once processing of the command data buffer has begun by the graphics hardware, the graphics driver returns from kernel mode to user mode to allow the graphics application to continue issuing commands. Depending on the textures loaded and the amount of texture memory, a ring transition is potentially required for every command data buffer. A Windows NT system creates an additional complexity by allowing multiple OpenGL applications to be active simultaneously. The necessity of ring transitions between user and kernal modes are relatively time consuming and reduce system bandwidth. Accordingly, a more efficient method of switching textures is desired.

Brief Summary Text (16):
According to a preferred embodiment, the invention provides a method for rendering textures in a graphics controller. The invention may be broadly conceptualized by the following steps: receiving draw commands from an application; forming a command data buffer responsive to the draw commands containing primitives corresponding to each draw command received from the application; forming a data structure responsive to the draw commands identifying a texture to be applied to the primitives; switching from user mode to kernel mode after forming the command data buffer and the data structure; in kernel mode, loading into the graphics controller a first texture identified in the data structure if the first texture is not present in the graphics controller; and in kernel mode, sending the primitives corresponding to the first texture to the graphics controller. The loading and sending steps may be repeated for each texture identified in the data structure.

Drawing Description Text (4):
FIG. 2 is a block diagram illustrating the graphics controller of FIG. 1, according to a preferred embodiment;

Detailed Description Text (2):
As shown in the drawings for purposes of illustration, the invention is embodied in a computer system S that includes a graphics controller for efficiently downloading texture maps into a frame buffer. The graphics controller is beneficially operable to process multiple texture commands without switching between user and privileged modes in a multi-tasking computer.

Detailed Description Text (3):
Referring first to FIG. 1, there is illustrated a computer system S according to a preferred embodiment. A central processing unit (CPU) 10 is coupled to a memory controller hub 12. The memory controller hub 12 is further coupled to a main memory 14, an input/output controller hub 16, and a graphics controller 18. The memory controller hub 12 also provides an interface to a peripheral component interface (PCI) bus 20. The graphics controller 18 is connected to the memory controller hub 12 by an accelerated graphics port (AGP), or AGP bus 22. Output of the graphics controller 18 is provided to a display output device 24 for displaying 2D and 3D graphics, text and video.

Detailed Description Text (5):
In the operation of the computer system S, a graphics application is executed on the CPU 10. The graphics application receives input from interaction devices such as the pointing device 26 or keyboard 28, and generates images for output on the display output device 24. The graphics controller 18 receives a series of graphics output commands from the application to develop the images that contain both a geometric description of what is to be displayed and the attributes describing how the objects should appear.

Detailed Description Text (6):
Referring next to FIG. 2, there is illustrated a high-level block diagram of the graphics controller 18 showing a graphics pipeline. Data is received by the graphics controller 18 from the CPU 10 over the AGP bus 22 into a two-dimensional (2D) engine 50 for handling 2D functions such as drawing primitives (points, lines, curves, circles, polygons, etc.) filling primitives, clipping primitives, and generating characters. The 2D engine 50 also includes a direct memory access (DMA) engine to quickly transfer data between main memory 14 and various buffers of the graphics controller 18, particularly a first-in first-out (FIFO) memory for receiving commands and data. Next in the graphics pipeline is one or more geometry accelerator engines 52 that receive commands and data from the CPU 10, via the 2D engine 50, for performing 3D geometrical transformations, such as translation, scaling, and rotation. In a preferred embodiment, up to six geometry accelerator engines 52 are used although other variations may use more than six for enhanced performance. Data from the 2D engine 50 and geometry accelerator engine 52 is provided to one or more rasterizer/texture (RT) engines 54 for performing rasterization, texture mapping and display scanning on rendered images which are stored in a frame buffer memory 56. In a preferred embodiment, one RT engine is used although other variations may use more than one for enhanced performance.

Detailed Description Text (7):
Now referring to FIG. 3, there is illustrated a block diagram showing a software/hardware architecture according to a preferred embodiment. In the operation of the computer system S, one or more graphics applications 60 are executed on the CPU 10 in user mode. Generally, a graphics application 60 receives input from interaction devices such as the pointing device 26 or keyboard 28, and generates images for output on the display output device 24. Each graphics application 60 communicates with an OpenGL driver interface 62 to pass graphics commands to the graphics driver. Each command is received by an OpenGL driver 64, which converts the command into low-level hardware commands which are [read] sent to the graphics controller 18 in kernel mode. In certain operating systems such as Windows NT, kernel mode allows the driver 64 access privileges to communicate directly with hardware whereas user mode does not.

Detailed Description Text (8):
In a preferred embodiment, texture maps are transferred to the graphics controller 18 with a DMA operation, although other data transfer techniques could be used. This requires the OpenGL driver 64 to communicate directly with hardware in kernel mode to initialize the DMA engine of the 2D engine 50. Since textures must be present in order to be applied during the execution of the texture commands, ring transitions into kernel mode to download textures is quite common.

Detailed Description Text (9):
Reference is directed to FIG. 4. In a preferred embodiment, the OpenGL driver 64 forms a data structure 70 and a command data (CD) buffer 72 in main memory 14 from the commands received from the graphics application 60. The data structure 70 and CD buffer specify the series of primitives. As the CD buffer 72 is developed in user mode, a texture ID 78 is placed in the data structure along with an offset 76 of the location in the CD buffer where the texture reference is needed. The data structure also contains an indicator 74 of how many texture operations will be performed. Essentially, the texture ID 78 is included as a placeholder for a texture map that may not yet have been loaded. Thus, the data structure 70 includes fields helpful in interpreting the CD buffer 72. The advantage of this technique is that the driver can build a CD buffer 72 without having the texture map pre-loaded. As a result, the OpenGL driver 64 can reference multiple textures while processing a single CD buffer (containing multiple draw operations).

Detailed Description Text (10):
Reference is now directed to FIGS. 5A and 5B where there is illustrated a flow diagram showing a method of avoiding processor ring transitions performed by the OpenGL driver 64 while rendering frames. In the prior art, when the active texture is changed this would force the OpenGL driver 64 to perform a ring transition to process the commands currently contained in the CD buffer 72. This is because certain operating systems like Windows NT only allow hardware access in kernel mode. Thus, given these constraints the OpenGL driver 64 would ordinarily transition to kernel mode to check whether the active texture has already been loaded. The present invention allows multiple texture changes to be accumulated at user level before making the ring transition to kernel mode to process the CD buffer 72. This amoratizes the cost of ring transition changes over several texture changes thereby improving performance of the graphics system.

Detailed Description Text (11):
A graphics application develops drawing commands which are received by the OpenGL driver 64 via the OpenGL API 62. At a step 100, the driver 64 begins to parse the commands to form a CD buffer 72. Registers in the RT engine 54 are initialized in accordance with the texture draw operation when the CD buffer 72 is first read. At step 102, the driver 64 determines whether a first command references a new texture map. If a new texture map is not referenced (meaning the command does not require a texture or the texture is currently the texture being referenced) the drawing command is written to the CD buffer 72 at step 104. If a new texture is referenced, the current offset in the CD buffer 72 is written to the data structure 70 along with a texture ID at step 106. The texture ID identifies the texture to be applied to the primitive 80 referenced by the offset.

Detailed Description Text (12):
Processing from steps 104 and 106 proceeds to step 108 where the OpenGL driver 64 determines whether the CD buffer 72 is full or whether the received commands from the graphics application 60 have been exhausted. In a preferred embodiment, the CD buffer 72 is artificially limited to thirty-two texture draw commands 74, but more or less than thirty-two could be used without departing from the principles of the invention. If the result is negative, processing branches from step 108 back to step 100 to receive another command from the graphics application 60. If the result is positive, the OpenGL driver 64 transitions from user mode to kernel mode and passes the data structure as an argument.

Detailed Description Text (14):
If at step 112, it is determined that no further texture transitions are contained in the CD buffer 72, processing branches to step 124 where the remaining contents of the CD buffer 72 are sent to hardware. The remaining contents could be primitives corresponding to the last loaded texture, or draw command not requiring a texture. At step 126, the driver 64 causes a ring transition to return to user mode to continue processing more commands from the graphics application 60.

CLAIMS:

1. A method of rendering textures in a graphics controller, comprising: receiving draw commands from an application; forming a command data buffer responsive to the draw commands containing primitives corresponding to each draw command received from the application; forming a data structure responsive to the draw commands identifying a texture to be applied to the primitives; switching from user mode to kernel mode after forming the command data buffer and the data structure; in kernel mode, loading into

the graphics controller a first texture identified in the data structure if the first texture is not present in the graphics controller; and in kernel mode, sending the primitives corresponding to the first texture to the graphics controller.

4. A method of rendering textures in a graphics controller, comprising: (a) storing draw commands in a command data buffer; (b) storing a texture ID and an offset in a data structure if a new texture is referenced while storing draw commands in the command data buffer; (c) switching from user mode to kernel mode; (d) in kernel mode, sending the draw commands to the graphics controller until an offset in the data structure is reached; (e) in kernel mode, loading into the graphics controller a texture identified in the data structure if the texture is not present in the graphics controller; (f) in kernel mode, sending the draw commands to the graphics controller from the offset until a next offset in the data structure is reached; (g) in kernel mode, repeating steps (e) and (f) until the command data buffer is exhausted; and (h) switching from kernel mode to user mode.

☐ 11. Document ID: US 6466998 B1

L3: Entry 11 of 23 File: USPT Oct 15, 2002

DOCUMENT-IDENTIFIER: US 6466998 B1
TITLE: Interrupt routing mechanism for routing interrupts from peripheral bus to interrupt controller

Detailed Description Text (3):
Attention now is directed to the drawings and particularly to FIG. 1, an example computer system platform having an APIC system such as Intel 82489DX APIC and an example interrupt controller such as Intel 8259 PIC incorporated to handle different levels of interrupts while ensuring proper operation of currently available operating systems (OS) according to the principles of the present invention is illustrated. As shown in FIG. 1, the computer system 100 may comprise a processor subsystem 110, a memory subsystem 120, connected to the processor subsystem 110 by a front side bus 10, graphics 130 connected to the memory subsystem 120 by a AGP or graphics bus 30, one or more host chipsets (e.g., expansion bridges) 140-150 connected to the memory subsystem 120 by hub links 40 and 50 for providing an interface with peripheral buses such as a non-legacy Peripheral Component Interconnect (PCI) buses 60 and 70 of different bandwidths and operating speeds, a flash memory 160, and a super I/O 170 connected to the chipset 150 by a low pin count (LPC) bus for providing an interface with a plurality of I/O devices 180, including, for example, a keyboard controller for controlling operations of an alphanumeric keyboard, a cursor control device such as a mouse, track ball, touch pad, joystick, etc., a mass storage device such as magnetic tapes, hard disk drives (HDD), and floppy disk drives (FDD), and serial and parallel ports to printers, scanners, and display devices. A different plurality of I/O devices 190 may be provided by the non-legacy PCI bus 60. In addition, it may be noted that the computer system 100 may be configured differently or employ some or different components than those shown in FIG. 1.

Detailed Description Text (4):
The processor subsystem 110 may include a plurality of host processors 110a-110nn. Each of the host processors 110a-110n may include a local APIC (not shown) of an APIC system such as an Intel 82489DX APIC described in the "MultiProcessor Specification (MPS)" Version 1.1., September 1994, Order Number 242016-003 from Intel Corporation. The memory subsystem 120 may include a memory controller hub (MCH) 122 connected to the host processors 110a-110n by a front side bus 10 (i.e., host bus or processor bus) and at least one memory element 124 connected to the MCH 122 by a memory bus 20. The memory element 124 may preferably be a dynamic random-access-memory (DRAM), but may be substituted for read-only-memory (ROM), video random-access-memory (VRAM) and the like. The memory element 124 stores information and instructions for use by the host processors 110a-110n. The graphics 130 may be connected to the main controller hub 122 of the memory subsystem 120 by an AGP bus (graphics bus) 30, and may include, for example, a graphics controller, a local memory and a display device (e.g., cathode ray

tube, liquid crystal display, flat panel display, etc.).

Detailed Description Text (12):
Interrupts may be generated by a number of different sources, including, for example: (1) external I/O devices 180 connected to 10 APIC 142 of P64H 140 manifested by either edges (level transitions) or levels on interrupt input pins and may be redirected to any processor; (2) locally connected device interrupts; (3) APIC timer interrupts; (4) inter-processor interrupts addressed to any individual processor of groups of processors in support of software self interrupts, pre-emptive scheduling, cache memory table look-aside buffer flushing, and interrupt forwarding; and (5) bus parity error interrupt. For the sake of simplicity and for purposes of this disclosure, the interrupts for routing to an interrupt controller 156 such as Intel 8259 PIC may be those generated from a non-legacy PCI bus 60. However, those interrupts may not be limited thereto. Both PCI and non-PCI interrupts received from external I/O devices may be utilized.

☐ 12. Document ID: US 6466226 B1

L3: Entry 12 of 23 File: USPT Oct 15, 2002

DOCUMENT-IDENTIFIER: US 6466226 B1
TITLE: Method and apparatus for pixel filtering using shared filter resource between overlay and texture mapping engines

Abstract Text (1):
A configurable filter module for providing shared filter resource between an overlay engine and a texture mapping engine of a graphics system. The configurable filter may comprise a plurality of linear blend units each of which receives data input from one of the overlay engine and a mapping engine cache, and generates a linear blend filter output respectively; and a filter output multiplexer which receives data output from the linear blend units and selects a proper byte ordering output, wherein the linear blend units serve as an overlay interpolator filter to perform linear blending of the data input from the overlay engine during a linear blend mode, and serve as a texture bilinear filter to perform bilinear filtering of the data input from the mapping engine cache during a bilinear filtering mode.

Brief Summary Text (2):
The present invention relates to computer graphics, and more particularly, relates to a method and apparatus for pixel filtering using commonly shared filter resource between an overlay engine (2D graphics engine) and a texture mapping engine (3D graphics engine) in a computer system.

Brief Summary Text (4):
A typical computer system includes a processor subsystem of one or more microprocessors such as Intel.RTM. i386, i486, Celeron.TM. or Pentium.RTM. processors, a memory subsystem, one or more chipsets provided to support different types of host processors for different platforms such as desktops, personal computers (PC), servers, workstations and mobile platforms, and to provide an interface with a plurality of input/output (I/O) devices including, for example, keyboards, input devices, disk controllers, and serial and parallel ports to printers, scanners and display devices. Chipsets may integrate a large amount of I/O bus interface circuitry and other circuitry onto only a few chips. Examples of such chipsets may include Intel.RTM. 430, 440 and 450 series chipsets, and more recently Intel.RTM. 810 and 8XX series chipsets. These chipsets may implement, for example, the I/O bus interface circuitry, direct memory access (DMA) controller, graphics controller, graphics memory controller, and other additional functionality such as graphics visual and texturing enhancements, data buffering, and integrated power management functions.

Brief Summary Text (5):
For graphics/multimedia applications, video data may be obtained from a video source by a graphics controller and displayed on a display monitor for viewing purposes. In

traditional three-dimensional (3D) graphics systems, 3D images may be generated for representation on a two-dimensional (2D) display monitor. The 2D representation may be provided by defining a 3D model space and assigning sections of the 3D model space to pixels for a visual display on the display monitor. Each pixel may display the combined visual effects such as color, shade and transparency defined on an image.

Brief Summary Text (6):
The visual characteristics of the 2D representation of the 3D image may also be enhanced by texturing. Texture may represent changes in intensity, color, opacity, or thematic contents (such as surface material type). The process of applying texture patterns to surfaces (adding graphics to scenery) is generally referred to as "texture mapping" and is well known and widely used technique in computer graphics. The texture may be represented by a 2D array of video data. Data elements in the array are called texels and the array is called a texture map. The two coordinate axes of the texture coordinate space are defined by rows and columns of the array typically designated in "U" and "V" coordinates.

Brief Summary Text (8):
However, separate 2D and 3D arithmetic circuits are necessarily required at separate locations (i.e., the overlay engine and the 3D engine) to perform the 2D overlay stretch blit and the 3D texture cache functions. These arithmetic circuits can be burdensome and cost-prohibitive. In addition, separate linear interpolators are also required for different data formats to calculate multiple color resolutions.

Brief Summary Text (9):
Accordingly, a need exists for a cost-effective filter solution with less hardware to eliminate the need to create separate 2D and 3D arithmetic circuits for the 2D overlay stretch blit and the 3D texture cache functions, and separate linear interpolators for different data formats for multiple color resolutions.

Brief Summary Text (11):
Accordingly, various embodiments of the present invention are directed to a configurable filter module for providing commonly shared filter resource between an overlay engine and a texture mapping engine of a graphics system. Such a filter module may comprise a plurality of linear blend units each of which receives data input from one of the overlay engine and a mapping engine cache, and generates a linear blend filter output respectively; a filter output multiplexer which receives data output from the linear blend units and selects a proper byte ordering output, wherein the linear blend units serve as an overlay interpolator filter to receive data input from the overlay engine for a linear blending function during a linear blend mode, and serve as a texture bilinear filter to receive data input from the mapping engine cache for a texture bilinear filtering function during a bilinear filtering mode.

Drawing Description Text (5):
FIG. 3 illustrates a functional diagram of an example graphics and memory controller hub (GMCH) according to an embodiment of the present invention;

Drawing Description Text (6):
FIG. 4 illustrates a top level I/O interconnect diagram of an example mapping engine cache output (MECO) unit for pixel filtering and providing shared filter resource functionality between an overlay engine and a 3D (texture mapping) engine according to an embodiment of the present invention;

Drawing Description Text (7):
FIG. 5 illustrates a block diagram of an example mapping engine cache output (MECO) unit for pixel filtering and providing shared filter resource functionality between an overlay engine and a 3D (texture mapping) engine according to an embodiment of the present invention;

Detailed Description Text (2):
The present invention is applicable for use with all types of computer systems, processors, video sources and chipsets, including follow-on chip designs which link together work stations such as computers, servers, peripherals, storage devices, and consumer electronics (CE) devices for audio and video communications. The video sources may include video storage media, video equipments and/or video consumer electronics (CE) devices. Examples of such consumer electronics (CE) devices may include digital video discs (DVD), audio compact discs (CD), videotapes, laser discs, CD-ROMs (read only memory), digital video cameras, digital still cameras, HD-TVs, satellite networks, cable networks, video cassette recorders (VCR), printers, scanners, imaging systems and

cellular systems and those CE devices which may become available as technology advances in the future. However, for the sake of simplicity, discussions will concentrate mainly on a computer system having a basic graphics/multimedia platform architecture of multi-media engines executing in parallel to deliver high performance video capabilities, although the scope of the present invention is not limited thereto. The term "graphics" may include, but may not be limited to, computer-generated images, symbols, visual representations of natural and/or synthetic objects and scenes, pictures and text.

Detailed Description Text (3):
Attention now is directed to the drawings and particularly to FIG. 1, an example computer system 100 having a graphics/multimedia platform of multi-media engines according to an embodiment of the present invention is illustrated. The computer system 100 (which can be a system commonly referred to as a personal computer or PC) may include one or more processors or central processing units (CPU) 110 such as Intel.RTM. i386, i486, Celeron.TM. or Pentium.RTM. processors, a memory controller 120 connected to the CPU 110 via a front side bus 10, a system memory 130 connected to the memory controller 120 via a memory bus 20, a graphics controller 15140 connected to the memory controller 120 via a graphics bus (e.g., Advanced Graphics Port "AGP" bus) 30.

Detailed Description Text (4):
Alternatively, the graphics controller 140 may also be configured to access the memory controller 120 via a peripheral bus such as a peripheral component interconnect (PCI) bus 40, if so desired. The PCI bus may be a high performance 32 or 64 bit synchronous bus with automatic configurability and multiplexed address, control and data lines as described in the latest version of "PCI Local Bus Specification, Revision 2.1" set forth by the PCI Special Interest Group (SIG) on Jun. 1, 1995 for added-on arrangements (e.g., expansion cards) with new video, networking, or disk memory storage capabilities. The graphics controller 140 controls a visual display of graphics and/or video images on a display monitor 150 (e.g., cathode ray tube, liquid crystal display and flat panel display). The display monitor 150 can be either an interlaced or progressive monitor, but typically is a progressive display device. A frame buffer 160 may be coupled to the graphics controller 140 for buffering the data from the graphics controller 140, CPU 110, or other devices within the computer system 100 for a visual display of video images on the display monitor 150.

Detailed Description Text (7):
A video stream decoder 180 is connected to the graphics controller 140 and receives the compressed video data stream from the DVD drive 170. The video stream decoder 180 buffers the compressed video data stream in a dynamic random access memory (DRAM) 190, which is coupled to the video stream decoder 180. Although a DRAM is preferred for the speed, other storage devices such as a read-only-memory (ROM) and video random-access-memory (VRAM) may be utilized for the memory 190. The video stream decoder 180 then retrieves the video data from the memory 190 as needed and decompresses and decodes the video data. The decoded video data is output to the graphics controller 140 for processing and eventual display on the display monitor 150.

Detailed Description Text (8):
In another embodiment of the present invention, the memory controller 120 and the graphics controller 140 can be integrated as a single graphics and memory controller hub (GMCH) including dedicated multi-media engines executing in parallel to deliver high performance 3D, 2D and motion compensation video capabilities. The GMCH may be implemented as a PCI chip such as, for example, PIIX4.RTM. chip and PIIX6.RTM. chip manufactured by Intel Corporation. In addition, such a GMCH may also be implemented as part of a host chipset along with an I/O controller hub (ICH) and a firmware hub (FWH) as described, for example, in Intel.RTM. 810 and 8XX series chipsets.

Detailed Description Text (9):
FIG. 2 illustrates an example computer system 100 including such a host chipset 200 according to an embodiment of the present invention. As shown in FIG. 2, the computer system 100 includes essentially the same components shown in FIG. 1, except for the host chipset 200 which provide a highly-integrated three-chip solution consisting of a graphics and memory controller hub (GMCH) 210, an input/output (I/O) controller hub (ICH) 220 and a firmware hub (FWH) 230.

Detailed Description Text (10):
The GMCH 210 provides graphics and video functions and interfaces one or more memory devices to the system bus 10. The GMCH 210 may include a memory controller as well as a

graphics controller (which in turn may include a 3D engine, a 2D engine, and a video engine). GMCH 210 may be interconnected to any of a system memory 130, a local display memory 155, a display monitor 150 (e.g., a computer monitor) and to a television (TV) via an encoder and a digital video output signal. GMCH 210 maybe, for example, an Intel.RTM. 82810 or 82810-DC100 chip. The GMCH 210 also operates as a bridge or interface for communications or signals sent between the processor 110 and one or more I/O devices which may be connected to an ICH 220.

Detailed Description Text (15):
FIG. 3 illustrates a block diagram of a graphics and memory controller hub (GMCH) 210 according to an example embodiment of the present invention. The GMCH 210 may include a graphics controller 140 to provide graphics and video functions and a memory controller 120 to control and interface one or more memory devices via the system bus 10. Memory controller 120 may be connected to the system bus 10 via a buffer 216 and a system bus interface 212. The memory controller 120 may also be connected to the ICH 220 via a buffer 216 and a hub interface 214. In addition, the GMCH 210 may be connected to a system memory 130 and, optionally, a local display memory 155 (also commonly referred to as video or graphics memory typically provided on a video card or video memory card). In a cost saving unified memory architecture (UMA), the local display memory 155 may be reside in the computer system. In such an architecture, the system memory 130 may operate as both system memory and the local display memory.

Detailed Description Text (16):
The graphics controller 140 of the GMCH 210 may include a 3D (texture mapping) engine 310 for performing a variety of 3D graphics functions, including creating a rasterized 2D display image from representation of 3D objects, a 2D engine 320 for performing 2D functions, a display engine 330 for displaying video or graphics images, and a digital video output port 340 for outputting digital video signals and providing connection to traditional TVs or new space-saving digital flat panel display.

Detailed Description Text (17):
The 3D (texture mapping) engine 310 performs a variety of functions including perspective-correct texture mapping to deliver 3D graphics without annoying visual anomalies such as warping, bending or swimming, bilinear and anisotropic filtering to provide smoother and more realistic appearance 3D images, MIP mapping to reduce blockiness and enhance image quality, Gouraud shading, alpha-blending, fogging and Z-buffering.

Detailed Description Text (19):
The display engine 330 includes a hardware motion compensation module 332 for performing motion compensation to improve video quality, a hardware cursor 334 for providing cursor patterns, an overlay engine 336 for merging either video data captured from a video source or data delivered from the 2D engine 320 with graphics data on the display monitor 150, and a digital-to-analog converter (DAC) 338 for converting digital video to analog video signals (YUV color space to RGB color space) for a visual display on the display monitor 150. The hardware motion compensation module 332 may alternatively reside within the 3D engine 310 for purposes of simplicity.

Detailed Description Text (21):
Turning now to FIG. 4, a top level I/O interconnect diagram of an example mapping engine cache output (MECO) unit 400 for pixel filtering and providing shared filter resource functionality between an overlay engine 336 and a 3D (texture mapping) engine 310 according to an embodiment of the present invention is illustrated. The MECO unit 400, a mapping engine cache 410 and a color calculator 420 may reside in the 3D engine 310 and form a texture pipeline within the 3D engine 310. The MECO unit 400 has an interface directly with the 2D engine 320 for receiving 2D inputs (64 bits A & B data input: pixels) from the 2D engine 320 through the overlay engine 336. The mapping engine cache 410 provides 3D inputs (16 bits A & B data input: texels) from the setup stage of the 3D (texture mapping) engine 310 to the MECO unit 400.

Detailed Description Text (22):
FIG. 5 illustrates a block diagram of an example mapping engine cache output (MECO) unit 400 for pixel filtering and providing time-domain shared filter resource functionality between an overlay engine and a 3D (texture mapping) engine according to an embodiment of the present invention. As shown in FIG. 5, the MECO unit 400 contains a shared filter module 500 for providing commonly shared filter resource functionality between the overlay engine 336 and the 3D (texture mapping) engine 310, and following downstream units, including, for example, a color space converter 510, an anisotropic filter module 520, a dithering unit 530, a re-order FIFO 540, and a motion compensation

module 550. The shared filter module 500 may be a re-configurable filter intended to serve as either an overlay interpolator (Overlay Vertical Stretch Blit) filter for linear blending 2D inputs from a 2D engine 320 through an overlay engine 336, or a bilinear texture filter for bilinear filtering 3D inputs from a 3D engine 310. The reconfigurable filter is designed to advantageously eliminate the need to create separate 2D and 3D arithmetic circuits for the 2D overlay stretch blit and the 3D texture cache functions. In either filter configuration, the shared filter module 400 may be utilized to bi-linear color values to approximate the perspective correct shading value of a 3D triangular surface and the vertical stretch blit in the 2D overlay. However, the shared filter module 500 can only service one module function at a time. Arbitration maybe required between the overlay engine 336 and the texture mapping engine cache 410 with overlay assigned the highest priority.

Detailed Description Text (27):
Based on the linear blend equation, a multiply-free linear blend unit (LBU) may be created. The optimization may be accomplished by noting that a binary multiplication can be achieved by summing the multiplicand by itself shifted by the bit position of any active bits (bits containing 1) in t he multiplier. In this situation, A will be selected when B is not. This allows selection of A or B as inputs to the master summer of the multiply, thus reducing the number of terms to 1/2 that would normally be required. In order to support two data formats, 565 pixel grouping (5 bits of red value, 6 bits of green value and 5 bits of blue value) or a 88 pixel grouping, the linear blend unit (LBU) may split into a three-bit multiply section and a five bit multiply section. Two of these split linear blend units may be combined into a dual linear blend unit (DLBU) with the capability of operating in an 88 resolution format or a 565 resolution format. Four such dual linear blend units (DLBU) plus one single linear blend unit (LBU) may be required for all pixel/texel formats. All filter modes may be controlled by filter inputs, such as an "OvalidIn" signal from the overlay engine 336 and a "565/88" filter mode select signal from the mapping engine cache (MEC) 410. The following modes of filtering are required: 1) overlay vertical interpolator filtering, and 2) bilinear texture filtering. In order to support all the precision needed by the downstream dithering unit 530 (see FIG. 5), the last dual linear blend unit (DLBU) may carry 24 bit precision out for RGB (eight bit precision for each R,G and B) dithering inputs. All other linear blend units (both dual and single) only carry eight bits of precision for 8 bit modes and split the precision to 565 for RGB.

Detailed Description Text (28):
FIG. 6 illustrates a block diagram of an example shared filter module 500 for providing commonly shared filter resource functionality between an overlay (2D) engine and a texture mapping (3D) engine according to an embodiment of the present invention. As shown in FIG. 6, the shared filter module 500 may comprise a plurality of linear blend units 610, 620, 630, 640 and 650 which receive 64 bit (2D) or 16 bit (3D) A & B data input from either an overlay engine 336 or a mapping engine cache (MEC) 410, and generate dual linear blend filter output respectively, via respective registers 612, 622, 632, and 642, and a filter output multiplexer 660 which receives data output from the linear blend units 610, 620, 630, 640 and 650 and selects the proper byte ordering for the downstream units, i.e., the color space converter 510 via registers 662, 664, 666 and 668. There may be nine linear blend units used to form the shared filter module 500 (four dual linear blend units (LBU0-LBU3) and a single linear blend unit (LBU8)). Each dual linear blend unit (LBU0-LBU3) is designed to support two data formats, for example, 565 and 88 configurations. Each dual linear blend unit (LBU0-LBU3) may be configured as two split linear blend units or three split linear blend units and the associated circuitry to support both data formats. Dual linear blend unit (LBU3) 640 may be arranged to receive 64 bit (2D) or 16 bit (3D) A & B data input via selectors 602 and 604 under control of a filter mode signal (bilinear/linear blend control bits). In contrast to the dual linear blend unit (LBU0-LBU3), the single linear blend unit (LBU8) supports only one data format, that is the 88 configuration. In either blend unit, rounding circuitry may be provided to round away from zero with signed data.

Detailed Description Text (30):
When operated in the bilinear mode, the shared filter module 500 serves as a texture bilinear filter to receive 3D input data from the mapping engine cache 410. 3D input data may consist of texels. Bilinear filtering may be accomplished on texels using the equation: C=C1(1.u)(1-.v) +C2(.u(1-.v))+C3 (.u*.v)+C4(1-.u)*v, where C1, C2,C3 and C4 are the four adjacent texels making up the locations U-V, U+1-V, U-V+1 and U+1-V+1. The values .u and .v are the fractional locations within the C1, C2, C3, C4 texels. Data formats supported for texels may include 1555 ARGB, 0565 ARGB and 4444 ARGB color formats, where A is alpha. Color spaces of YUV and RGB are also supported. Texel 1555, 565 and 4444 produce one 16 bit quantity (i.e. 1555, 565 or 4444) per clock.

Detailed Description Text (32):
Control bits (bilinear/linear blend) determine the filter configuration of the shared filter module 500. An overlay valid data signal may be used to control the arbitration and selection of the filter owner (i.e., overlay and texture). Arbitration may be performed between the overlay stretch blit and the texture cache functions. The overlay maintains the highest priority and the texture cache may be assigned the lowest (two state priority). When valid overlay is present (determined by the overlay valid signal) the overlay engine 336 owns the filter operation of the shared filter module 500 until the overlay valid signal is no longer asserted. During this time the texture pipeline within the 3D engine 310 freezes (if any) current operations and waits until the overlay engine 336 has completed use of the shared filter module 500.

Detailed Description Text (33):
FIG. 7 illustrates a filter configuration diagram of an example shared filter module 500 when configured for bilinear filter operation in Texel 1555 mode and Texel 4444 mode according to an embodiment of the present invention. As shown in FIG. 7, each of the dual linear blend units (LUB0-3) of the shared filter module 500 may be configured as three linear blend units 610A-610C, 620A-620C, and 640A-640C, and two linear blend units 630A-630B. Dual linear blend units (BLU0, BLU1 and BLU3) 610A-610C, 620A-620C and 640A-640C are configured for bi-linear filtering of A & B data input from the mapping engine cache 410 to approximate perspective correct shading value of a 3D triangular surface for a 565 resolution format. In contrast to BLU0, BLU1 and BLU3, the dual linear blend unit (LBU2) 630A-630B and the single LBU 650 are configured for bi-linear filtering of A & B data input from the mapping engine cache 410 to approximate perspective correct shading value of a 3D triangular surface for a 88 resolution format. Registers 710-780 may be provided to control operation of the filter configuration. As for Texel 565 mode, the dual linear blend units (BLU0, BLU1 and BLU3) 610A-610C, 620A-620C and 640A-640C are configured for bi-linear filtering of A & B data input from the mapping engine cache 410 to approximate perspective correct shading value of a 3D triangular surface for a 565 resolution format. However, the dual LUB2630A-630B and the single LBU 650 may not be used.

Detailed Description Text (36):
FIG. 10 illustrates a filter configuration diagram of an example shared filter when configured for operation in Overlay YUV 4:2:0/4:2:2 mode according to an embodiment of the present invention. YUV 4:2:0 is a planar format typically used for digital playback since planar YUV 4:2:0 format requires less bandwidth. In contrast to YUV 4:2:0, YUV 4:2:2 is a packed or interleaved format used for graphics generation and video processing since YUV 4:2:2 format provides a more detailed, richer display. As shown in FIG. 10, all dual linear blend units (LUB0-3) of the shared filter module 500 may be configured as two linear blend units 610A-610B, 620A-620B, 630A-630B and 640A-640B for bi-linear filtering of A & B data input from the mapping engine cache 410 to approximate perspective correct shading value of a 3D triangular surface for a 88 resolution format. Registers 930-940 may be provided to control operation of the filter configuration.

Detailed Description Text (39):
As described from the foregoing, the present invention advantageously provides a shared filter module designed with minimal hardware for providing commonly shared filter resource between an overlay engine and a 3D (texture mapping) engine in order to eliminate the need to create separate 2D and 3D arithmetic circuits for the 2D overlay stretch blit and the 3D texture cache functions, and separate linear interpolators for different data formats for multiple color resolutions.

CLAIMS:

1. A configurable filter module for providing shared filter resource between an overlay engine and a texture mapping engine of a graphics system, comprising: a plurality of linear blend units to receive data input from one of an overlay engine and a mapping engine cache; and a filter output multiplexer to receive data output from the linear blend units and select a proper byte ordering output, wherein said linear blend units serve as an overlay interpolator filter to perform linear blending of the data input from the overlay engine during a linear blend mode, and serve as a texture bilinear filter to perform bilinear filtering of the data input from the mapping engine cache during a bilinear filtering mode.

5. The configurable filter module as claimed in claim 3, wherein said bilinear filtering is accomplished on texels using the equation:

C=C1(1-.u)(1-.v)+C2(.u(1-.v))+C3(.u*.v)+C4(1.u)*.v, where C1, C2,C3 and C4 represent 3-dimentional texel data from the mapping engine cache indicating four adjacent texels of locations U-V, U+1-V, U-V+1 and U+1-V+1, and where values .u and .v indicate fractional locations within the C1, C2, C3, C4 texels.

6. The configurable filter module as claimed in claim 3, wherein requests from the overlay engine for overlay interpolation take precedence over requests from the mapping engine cache.

7. The configurable filter module as claimed in claim 1, wherein said linear blend units can be configured as one of eight 8-bit linear interpolators, three 8-bit bi-linear interpolators and four 565 bi-linear interpolators to perform either said linear blending or said bilinear filtering of data input from respective overlay engine and mapping engine cache.

8. The configurable filter module as claimed in claim 1, wherein said linear blend units are configured as a combination of three thrice-split linear blend units, a twice-split linear blend unit and a single linear blend unit for bilinear filtering data input from the mapping engine cache to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

10. The configurable filter module as claimed in claim 1, wherein said linear blend units are configured as a combination of four dual linear blend units and a single linear blend unit arranged in parallel for bilinear filtering data input from the mapping engine cache to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

12. A graphics controller for processing video data comprising: a 3D engine which performs 3D graphics functions, including creating a rasterized 2D display image from representation of 3D; a 2D engine which performs 2D graphics functions and includes a blitter (BLT) engine and an arithmetic stretch blitter (BLT) engine for performing fixed blitter and stretch blitter (BLT) operations; and a display engine which enables a visual display of video or graphics images and includes an overlay engine for merging either video data captured from a video source or data delivered from the 2D engine with graphics data for said visual display; wherein said 3D engine comprises: a mapping engine cache which provides 3D data, a configurable filter which shares filter resource with said 2D engine via said overlay engine to perform either linear blending of 2D data input from said 2D engine via said overlay engine during a linear blend mode or bilinear filtering of 3D data input from said mapping engine cache during a bilinear filtering mode, and a color calculator which handles color calculations of texel maps for said visual display.

13. The graphics controller as claimed in claim 12, wherein said 3D engine also performs a variety of functions including perspective-correct texture mapping to deliver 3D graphics, bilinear and anisotropic filtering, MIP mapping to reduce blockiness and enhance image quality, Gouraud shading, alpha-blending, fogging and Z-buffering.

14. The graphics controller as claimed in claim 12, wherein said display engine further comprises: a hardware motion compensation module which performs motion compensation; a hardware cursor which provides cursor patterns; and a digital-to-analog converter (DAC) which converts digital video to analog video signals for said visual display on a display monitor.

15. The graphics controller as claimed in claim 12, wherein said 3D engine further comprises: a color space converter which receives YUV data and converts into RGB data, where YUV represents color-difference video data containing one luminance component (Y) and two chrominance components (U, V), and RGB represents composite video data containing red (R), green (G) and blue (B) components of an image; an anisotropic filter which sums four pixels from different levels-of-detail (LOD) levels to produce an average of four LOD levels; a dithering unit which reads dither weights from a table and sums the dither weights with the current pixel data received from the anisotropic filter; a re-ordering FIFO which sorts pixels for the proper output format; and a motion compensation unit which averages successive pixels, sums an error term with the averaged result, and sends data to the color calculator for handling final color calculations before rendering on said display monitor.

16. The graphics controller as claimed in claim 12, wherein said configurable filter comprises: a plurality of linear blend units each of which receives data input from one

of said overlay engine and said mapping engine cache, and generates a linear blend filter output respectively; and a filter output multiplexer which receives data output from the linear blend units and selects a proper byte ordering output, wherein said linear blend units serve as an overlay interpolator filter to perform said linear blending of the data input from said overlay engine during a linear blend mode, and serve as a texture bilinear filter to perform said bilinear filtering of the data input from said mapping engine cache during a bilinear filtering mode.

17. The graphics controller as claimed in claim 16, wherein said plurality of linear blending units comprise four dual linear blend units provided to support at least two data formats, and a single linear blend unit provided to support only one data format.

18. The graphics controller as claimed in claim 16, wherein said dual linear blend units are configured as either two split linear blend units or three split linear blend units and include associated circuitry to support both data formats under control of a filter select signal.

19. The graphics controller as claimed in claim 16, wherein said linear blending is accomplished on pixels using the equation A+alpha(B-A), where A represents 2-dimensional pixel data from the overlay engine indicating overlay surface A, B represents 2-dimensional data from the overlay engine indicating overlay surface B, and alpha represents a blending coefficient.

20. The graphics controller as claimed in claim 16, wherein said bilinear filtering is accomplished on texels using the equation:
C=C1(1-.u)(1-.v)+C2(.u(1-.v))+C3(.u*.v)+C4(1-.u) *.v, where C1, C2,C3 and C4 represent 3-dimentional texel data from the mapping engine cache indicating four adjacent texels of locations U-V, U+1-V, U-V+1 and U+1-V+1, and where values .u and .v indicate fractional locations within the C1, C2, C3, C4 texels.

21. The graphics controller as claimed in claim 16, wherein requests from said overlay engine for overlay interpolation take precedence over requests from said mapping engine cache.

22. The graphics controller as claimed in claim 16, wherein said linear blend units can be configured as one of eight 8-bit linear interpolators, three 8-bit bi-linear interpolators and four 565 bi-linear interpolators to perform either said linear blending or said bilinear filtering of data input from respective overlay engine and mapping engine cache.

23. The graphics controller as claimed in claim 16, wherein said linear blend units are configured as a combination of three thrice-split linear blend units, a twice-split linear blend unit and a single linear blend unit for bilinear filtering data input from said mapping engine cache to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

24. The graphics controller as claimed in claim 16, wherein said linear blend units are configured as four thrice-split linear blend units arranged in parallel for linear blending data input from said overlay engine to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

25. The graphics controller as claimed in claim 16, wherein said linear blend units are configured as a combination of four dual linear blend units and a single linear blend unit arranged in parallel for bilinear filtering data input from said mapping engine cache to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

26. The graphics controller as claimed in claim 16, wherein each of said linear blend units act as a single interpolator to calculate multiple color resolutions of different data format precision, and comprises: a high order 5-bit calculation unit arranged to shift data input from left to right by three bit positions; a high order 3-bit calculation unit arranged to shift data input from left to right by five bit positions; first adders arranged to add outputs from the high order 5-bit and 3-bit calculation units to create a high order 8-bit precision calculation; a low order 3-bit calculation unit arranged to shift data input from left to right by five bit positions; a low order 5-bit calculation unit arranged to shift data input from left to right by three bit positions; second adders arranged to add outputs from the low order 5-bit and 3-bit calculation units to create a low order 8-bit precision calculation; and means for calculating multple color resolutions of different data format precision based on said

high order 8-bit precision calculation and said low order 8-bit precision calculation.

27. A method for providing shared filter functionality between an overlay engine and a texture mapping engine in a graphics system to process video data comprising: receiving video data from one of said overlay engine and said texture mapping engine; configuring a plurality of linear blend units as either an overlay interpolator filter to perform linear blending of said video data input from said overlay engine or a bilinear texture filter to perform bilinear filtering of said video data input from said texture mapping engine; and determining filter color values to approximate perspective shading of a 3D triangular surface of an image in different resolution formats.

35. A graphics controller including a two-dimensional (2D) engine and a three-dimensional (3D) engine for processing data for a visual display, comprising: an overlay engine to provide 2D data from the 2D engine; a mapping engine cache to provide 3D data from the 3D engine; and a configurable filter to provide shared filter resources and to perform linear blending of 2D data from said 2D engine via said overlay engine, or bilinear filtering of 3D data from said mapping engine cache for subsequent visual display.

36. The graphics controller as claimed in claim 35, wherein: said 2D engine performs 2D graphics functions and includes a blitter (BLT) engine and an arithmetic stretch blitter (BLT) engine for performing fixed blitter and stretch blitter (BLT) operations; and said 3D engine performs 3D graphics functions, including creating a rasterized 2D display image from representation of 3D.

37. The graphics controller as claimed in claim 36, wherein said 3D engine also performs a variety of functions including perspective-correct texture mapping to deliver 3D graphics, bilinear and anisotropic filtering, MIP mapping to reduce blockiness and enhance image quality, Gouraud shading, alpha-blending, fogging and Z-buffering.

38. The graphics controller as claimed in claim 35, wherein said overlay engine is included in a display engine for merging either data captured from a video source or data delivered from said 2D engine with graphics for said visual display, said display engine further comprising: a hardware motion compensation module to perform motion compensation; a hardware cursor to provide cursor patterns; and a digital-to-analog converter (DAC) to convert digital video to analog video signals for said visual display on a display monitor.

39. The graphics controller as claimed in claim 36, wherein said 3D engine further comprises: a color space converter to receive YUV data and convert into RGB data, where YUV represents color-difference video data containing one luminance component (Y) and two chrominance components (U, V), and RGB represents composite video data containing red (R), green (G) and blue (B) components of an image; an anisotropic filter to combine pixels from different levels-of-detail (LOD) levels to produce an average of four LOD levels; a dithering unit to read dither weights from a table and sum the dither weights with the current pixel data received from the anisotropic filter; a re-ordering FIFO to sort pixels for the proper output format; and a motion compensation unit to average successive pixels, sum an error term with the averaged result, and send data for final color calculations before rendering on said display monitor.

40. The graphics controller as claimed in claim 36, wherein said configurable filter comprises: a plurality of linear blend units to receive data input from one of said overlay engine and said mapping engine cache; and a filter output multiplexer to receive data output from the linear blend units and select a proper byte ordering output, wherein said linear blend units serve as an overlay interpolator filter to perform said linear blending of the data input from said overlay engine during a linear blend mode, and serve as a texture bilinear filter to perform said bilinear filtering of the data input from said mapping engine cache during a bilinear filtering mode.

41. The graphics controller as claimed in claim 40, wherein said plurality of linear blending units comprise four dual linear blend units provided to support at least two data formats, and a single linear blend unit provided to support only one data format.

42. The graphics controller as claimed in claim 40, wherein said dual linear blend units are configured as either two split linear blend units or three split linear blend units and include associated circuitry to support both data formats under control of a filter select signal.

43. The graphics controller as claimed in claim 40, wherein said linear blending is accomplished on pixels using the equation A+alpha(B-A), where A represents 2-dimensional pixel data from the overlay engine indicating overlay surface A, B represents 2-dimensional data from the overlay engine indicating overlay surface B, and alpha represents a blending coefficient.

44. The graphics controller as claimed in claim 40, wherein said bilinear filtering is accomplished on texels using the equation: C=C1(1-.u)(1-.v)+C2(.u(1-.v))+C3(.u*.v)+C4(1-.u)*.v, where C1, C2,C3 and C4 represent 3-dimensional texel data from the mapping engine cache indicating four adjacent texels of locations U-V, U+1-V, U-V+1 and U+1-V+1, and where values .u and .v indicate fractional locations within the C1, C2, C3, C4 texels.

45. The graphics controller as claimed in claim 40, wherein requests from said overlay engine for overlay interpolation take precedence over requests from said mapping engine cache.

46. The graphics controller as claimed in claim 40, wherein said linear blend units can be configured as one of eight 8-bit linear interpolators, three 8-bit bi-linear interpolators and four 565 bi-linear interpolators to perform either said linear blending or said bilinear filtering of data input from respective overlay engine and mapping engine cache.

47. The graphics controller as claimed in claim 40, wherein said linear blend units are configured as a combination of three thrice-split linear blend units, a twice-split linear blend unit and a single linear blend unit for bilinear filtering data input from said mapping engine cache to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

48. The graphics controller as claimed in claim 40, wherein said linear blend units are configured as four thrice-split linear blend units arranged in parallel for linear blending data input from said overlay engine to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

49. The graphics controller as claimed in claim 40, wherein said linear blend units are configured as a combination of four dual linear blend units and a single linear blend unit arranged in parallel for bilinear filtering data input from said mapping engine cache to approximate perspective correct shading value of a 3-dimensional triangular surface for different resolution formats.

50. The graphics controller as claimed in claim 40, wherein each of said linear blend units act as a single interpolator to calculate multiple color resolutions of different data format precision, and comprises: a high order 5-bit calculation unit arranged to shift data input from left to right by three bit positions; a high order 3-bit calculation unit arranged to shift data input from left to right by five bit positions; first adders arranged to add outputs from the high order 5-bit and 3-bit calculation units to create a high order 8-bit precision calculation; a low order 3-bit calculation unit arranged to shift data input from left to right by five bit positions; a low order 5-bit calculation unit arranged to shift data input from left to right by three bit positions; second adders arranged to add outputs from the low order 5-bit and 3-bit calculation units to create a low order 8-bit precision calculation; and means for calculating multiple color resolutions of different data format precision based on said high order 8-bit precision calculation and said low order 8-bit precision calculation.

☐ 13. Document ID: US 6463004 B2

L3: Entry 13 of 23 File: USPT Oct 8, 2002

DOCUMENT-IDENTIFIER: US 6463004 B2
TITLE: VPX bank architecture

Detailed Description Text (7):
Referring now to FIG. 2, there is a computer system 200 that includes the present embodiment. Sample system 200 may have a memory incorporating a VPX banked memory architecture, in accordance with the present invention, such as in the embodiment described herein. Sample system 200 is representative of processing systems based on the PENTIUM.RTM., PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 200 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Washington, although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (9):
System 200 includes a memory 220. Memory 220 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 220 may store instructions and/or data represented by data signals that may be executed by processor 202. A cache memory 204 can reside inside processor 202 that stores data signals stored in memory 220. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (10):
A system logic chip 216 is coupled to the processor bus 210 and memory 220. The system logic chip 216 in the illustrated embodiment is a memory controller hub (MCH). The processor 202 communicates to a memory controller hub (MCH) 216 via a processor bus 210. The MCH 216 provides a high bandwidth memory path 218 to memory 220 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 216 directs data signals between processor 202, memory 220, and other components in the system 200 and bridges the data signals between processor bus 210, memory 220, and system I/O 222. In some embodiments, the system logic chip 216 provides a graphics port for coupling to a graphics controller 212. The MCH 216 is coupled to memory 220 through a memory interface 218. The graphics card 212 is coupled to the MCH 216 through an Accelerated Graphics Port (AGP) interconnect 214.

☐ 14. Document ID: US 6459645 B2

L3: Entry 14 of 23 File: USPT Oct 1, 2002

DOCUMENT-IDENTIFIER: US 6459645 B2
TITLE: VPX bank architecture

Detailed Description Text (7):
Referring now to FIG. 2, there is a computer system 200 that includes the present embodiment. Sample system 200 may have a memory incorporating a VPX banked memory architecture, in accordance with the present invention, such as in the embodiment described herein. Sample system 200 is representative of processing systems based on the PENTIUM.RTM., PENTIUM.RTM. Pro, PENTIUM.RTM.II, PENTIUM.RTM. III microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 200 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (9):
System 200 includes a memory 220. Memory 220 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 220 may store instructions and/or data represented by data signals that may be executed by processor 202. A cache memory 204 can reside inside

processor 202 that stores data signals stored in memory 220. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (10):
A system logic chip 216 is coupled to the processor bus 210 and memory 220. The system logic chip 216 in the illustrated embodiment is a memory controller hub (MCH). The processor 202 communicates to a memory controller hub (MCH) 216 via a processor bus 210. The MCH 216 provides a high bandwidth memory path 218 to memory 220 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 216 directs data signals between processor 202, memory 220, and other components in the system 200 and bridges the data signals between processor bus 210, memory 220, and system 222. In some embodiments, the system logic chip 216 provides a graphics port for coupling to a graphics controller 212. The MCH 216 is coupled to memory 220 through a memory interface 218. The graphics card 212 is coupled to the MCH 216 through an Accelerated Graphics Port (AGP) interconnect 214.

☐ 15. Document ID: US 6452438 B1

L3: Entry 15 of 23 File: USPT Sep 17, 2002

DOCUMENT-IDENTIFIER: US 6452438 B1
TITLE: Triple well no body effect negative charge pump

Detailed Description Text (5):
Referring now to FIG. 1, an exemplary computer system 100 is shown. System 100 includes a component, such as a processor, employing a triple well no body effect negative charge pump in accordance with the present invention, such as in the embodiment described herein. System 100 is representative of processing systems based on the PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif. although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 100 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Washington, although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (8):
System 100 includes a memory 120. Memory 120 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102. A cache memory 104 can reside inside processor 102 that stores data signals stored in memory 120. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (10):
The system logic chip 116 in the illustrated embodiment is a memory controller hub (MCII). The processor 102 communicates to the MCH 116 via a processor bus 110. The MCH 116 provides a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 116 directs data signals between processor 102, memory 120, and other components in the system 100 and bridges the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 provides a graphics port for coupling to a graphics controller 112. The MCH 116 is coupled to memory 120 through a memory interface 118. The graphics card 112 is coupled to the MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

Detailed Description Text (12):
A triple well no body effect negative charge pump 126 also resides in flash memory BIOS 128. Alternate embodiments of a triple well no body effect negative charge pump 106 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and

other types of logic circuits.

Detailed Description Text (13):
For another embodiment of a system, one implementation of a charge pump can be used with a system on a chip. One embodiment of a system on a chip comprises of a processor and a memory. The memory for one system is a flash memory. The flash memory can be located on the same die as the processor and other system components. Additionally, other logic blocks such as a memory controller or graphics controller can also be located on a system on a chip. By including one embodiment of the present invention on the system on a chip, the flash memory can be enabled to program and erase flash memory cells without requiring a high voltage pin on the system on a chip pin-out. The needed high voltage potentials can be generated on the same die.

☐ 16. Document ID: US 6441678 B1

L3: Entry 16 of 23 File: USPT Aug 27, 2002

DOCUMENT-IDENTIFIER: US 6441678 B1
TITLE: Self initialization forcharge pumps

Detailed Description Text (5):
Referring now to FIG. 1, an exemplary computer system 100 is shown. System 100 includes a component, such as a processor, employing self initialization for charge pumps in accordance with the present invention, such as in the embodiment described herein. System 100 is representative of processing systems based on the PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 100 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (8):
System 100 includes a memory 120. Memory 120 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102. A cache memory 104 can reside inside processor 102 that stores data signals stored in memory 120. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (9):
A system logic chip 116 is coupled to the processor bus 110 and memory 120. The system logic chip 116 in the illustrated embodiment is a memory controller hub (MCH). The processor 102 communicates to the MCH 116 via a processor bus 110. The MCH 116 provides a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 116 directs data signals between processor 102, memory 120, and other components in the system 100 and bridges the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 provides a graphics port for coupling to a graphics controller 112. The MCH 116 is coupled to memory 120 through a memory interface 118. The graphics card 112 is coupled to the MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

Detailed Description Text (11):
In one embodiment, a charge pump self initialization mechanism 106 resides in flash memory BIOS 128. Alternate embodiments of a self initialization mechanism 126 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits.

Detailed Description Text (12):
For another embodiment of a system, one implementation of a charge pump self initialization mechanism can be used with a system on a chip. One embodiment of a system on a chip comprises of a processor and a memory. The memory for one such system is a flash memory. The flash memory can be located on the same die as the processor and other system components. Additionally, other logic blocks such as a memory controller or graphics controller can also be located on a system on a chip. By including one embodiment of the present invention on the system on a chip, the charge pumps can self initialize and lower power consumption. The flash memory can be enabled to program and erase flash memory cells without requiring a high voltage pin on the system on a chip pin-out. The needed high voltage potentials can be generated on the same die.

☐ 17. Document ID: US 6434073 B1

L3: Entry 17 of 23 File: USPT Aug 13, 2002

DOCUMENT-IDENTIFIER: US 6434073 B1
TITLE: VPX bank architecture

Detailed Description Text (7):
Referring now to FIG. 2, there is a computer system 200 that includes the present embodiment. Sample system 200 may have a memory incorporating a VPX banked memory architecture, in accordance with the present invention, such as in the embodiment described herein. Sample system 200 is representative of processing systems based on the PENTIUM.RTM., PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 200 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (9):
System 200 includes a memory 220. Memory 220 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 220 may store instructions and/or data represented by data signals that may be executed by processor 202. A cache memory 204 can reside inside processor 202 that stores data signals stored in memory 220. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (10):
A system logic chip 216 is coupled to the processor bus 210 and memory 220. The system logic chip 216 in the illustrated embodiment is a memory controller hub (MCH). The processor 202 communicates to a memory controller hub (MCH) 216 via a processor bus 210. The MCH 216 provides a high bandwidth memory path 218 to memory 220 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 216 directs data signals between processor 202, memory 220, and other components in the system 200 and bridges the data signals between processor bus 210, memory 220, and system I/O 222. In some embodiments, the system logic chip 216 provides a graphics port for coupling to a graphics controller 212. The MCH 216 is coupled to memory 220 through a memory interface 218. The graphics card 212 is coupled to the MCH 216 through an Accelerated Graphics Port (AGP) interconnect 214.

☐ 18. Document ID: US 6377121 B1

L3: Entry 18 of 23 File: USPT Apr 23, 2002

DOCUMENT-IDENTIFIER: US 6377121 B1
TITLE: Dynamic cascoding technique for operational amplifiers

Detailed Description Text (7):
Referring now to FIG. 1, a computer system 100 is shown. System 100 includes a component, such as a flash memory 128, employing an operational amplifier (op amp) with dynamic cascodes 126 in accordance with the present invention, such as in the embodiment described herein. System 100 is representative of processing systems based on the Intel PENTIUM.RTM. II, PENTIUM.RTM. III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes, handheld telephones and the like) may also be used. In one embodiment, sample system 100 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Washington, although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (9):
System 100 includes a memory 120. Memory 120 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102. A cache memory 104 can reside inside processor 102 that stores data signals stored in memory 120. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (10):
A system logic chip 116 is coupled to the processor bus 110 and memory 120. The system logic chip 116 in the illustrated embodiment is a memory controller hub (MCH). The processor 102 communicates to the MCH 116 via a processor bus 110. The MCH 116 provides a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 116 directs data signals between processor 102, memory 120, and other components in the system 100 and bridges the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 provides a graphics port for coupling to a graphics controller 112. The MCH 116 is coupled to memory 120 through a memory interface 118. The graphics card 112 is coupled to the MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

Detailed Description Text (12):
The firmware hub in this system employs a flash memory device 128. An op amp with dynamic cascodes 126 to also resides in flash memory 128. Alternate embodiments of an op amp 126 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits. Op amps can also be referred to as amplifiers or buffers.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 19. Document ID: US 6374317 B1

L3: Entry 19 of 23 File: USPT Apr 16, 2002

DOCUMENT-IDENTIFIER: US 6374317 B1
TITLE: Method and apparatus for initializing a computer interface

Brief Summary Text (5):
Busses such as the PCI bus also provide for communication with other computer system

devices such as graphics controllers and network adapters. Because busses such as the PCI bus must interface with a variety of component types, each with varying requirements, the busses are not necessarily optimized for allowing communication between chipset components. Further, chipset manufacturers who rely on standardized busses such as the PCI bus must adhere to bus standards in order to ensure compatibility with other components, and are not at liberty to make substantial changes in how the chipset components communicate with each other.

Brief Summary Text (8):
According to one embodiment, a system is disclosed that includes a central processing unit (CPU) and a memory controller hub (MCH) coupled to the CPU. The MCH includes a first interface controller that is operable to detect the presence of a hub agent coupled to the MCH.

Detailed Description Text (10):
MCH 110 may also include a graphics interface 113 coupled to a graphics accelerator 130. In one embodiment, graphics interface 113 is coupled to graphics accelerator 130 via an accelerated graphics port (AGP) that operates according to an AGP Specification Revision 2.0 interface developed by Intel Corporation of Santa Clara, Calif. In other embodiments, graphics interface 113 may be implemented using a hub interface controller 120 coupled to graphics accelerator 130.

Detailed Description Text (13):
ICH 140 may also include a bridge 146 that provides a conventional interface to a PCI bus. Bridge 146 provides a data path between CPU 102 and peripheral devices. Devices that may be coupled to PCI bus 142 include an audio device 150 and a disk drive 155. However, one of ordinary skill in the art will appreciate that other devices may be coupled to PCI bus 142. In addition, one of ordinary skill in the art will recognize that CPU 102 and MCH 110 could be combined to form a single chip. Further graphics accelerator 130 may be included within MCH 110 in other embodiments.

Detailed Description Text (46):
In alternative embodiments, additional transaction attributes may include the ability to differentiate between "snooped" traffic where cache coherency is enforced by hardware (i.e., chipset) and "non-snooped" traffic that relies on software mechanisms to ensure data coherency in the system. Moreover, another possible attribute would be an "explicitly prefetchable" hint, to support a form of read caching and allow for more efficient use of the main memory bandwidth.

CLAIMS:

1. A computer system comprising:

a memory controller hub (MC); and

a first interface coupled to the MCH to transfer data directly between the MCH and a first hub agent within the computer system, the first interface comprising:

a data signal path to transmit data in packets via split transactions; and

set of command signals, the first interface providing a point-to-point connection between the MCH and the first hub agent, exclusive of an external bus connected directly to the first interface; the MCH operable to detect the presence of the first hub agent via the first interface.

5. The computer system of claim 3 wherein the first hub agent is a network interface card and the second hub agent is a graphics accelerator.

11. The computer system of claim 10 wherein the MCH further comprises:

a memory controller coupled to the CPU; and

a graphics interface.

☐ 20. Document ID: US 6366158 B1

L3: Entry 20 of 23 File: USPT Apr 2, 2002

DOCUMENT-IDENTIFIER: US 6366158 B1
TITLE: Self initialization for charge pumps

Detailed Description Text (5):
Referring now to FIG. 1, an exemplary computer system 100 is shown. System 100 includes a component, such as a processor, employing self initialization for charge pumps in accordance with the present invention, such as in the embodiment described herein. System 100 is representative of processing systems based on the PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 100 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (8):
System 100 includes a memory 120. Memory 120 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 120 may store instructions and/or data represented by data signals that may be executed by processor 102. A cache memory 104 can reside inside processor 102 that stores data signals stored in memory 120. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (9):
A system logic chip 116 is coupled to the processor bus 110 and memory 120. The system logic chip 116 in the illustrated embodiment is a memory controller hub (MCH). The processor 102 communicates to the MCH 116 via a processor bus 110. The MCH 116 provides a high bandwidth memory path 118 to memory 120 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 116 directs data signals between processor 102, memory 120, and other components in the system 100 and bridges the data signals between processor bus 110, memory 120, and system I/O 122. In some embodiments, the system logic chip 116 provides a graphics port for coupling to a graphics controller 112. The MCH 116 is coupled to memory 120 through a memory interface 118. The graphics card 112 is coupled to the MCH 116 through an Accelerated Graphics Port (AGP) interconnect 114.

Detailed Description Text (11):
In one embodiment, a charge pump self initialization mechanism 106 resides in flash memory BIOS 128. Alternate embodiments of a self initialization mechanism 126 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits.

Detailed Description Text (12):
For another embodiment of a system, one implementation of a charge pump self initialization mechanism can be used with a system on a chip. One embodiment of a system on a chip comprises of a processor and a memory. The memory for one such system is a flash memory. The flash memory can be located on the same die as the processor and other system components. Additionally, other logic blocks such as a memory controller or graphics controller can also be located on a system on a chip. By including one embodiment of the present invention on the system on a chip, the charge pumps can self initialize and lower power consumption. The flash memory can be enabled to program and erase flash memory cells without requiring a high voltage pin on the system on a chip pin-out. The needed high voltage potentials can be generated on the same die.

☐ 21. Document ID: US 6356105 B1

L3: Entry 21 of 23 File: USPT Mar 12, 2002

DOCUMENT-IDENTIFIER: US 6356105 B1
TITLE: Impedance control system for a center tapped termination bus

Detailed Description Text (6):
Referring now to FIG. 3, a computer system 300 is shown. System 300 includes a component, such as a memory controller hub 316, employing an impedance control mechanism in accordance with the present invention, such as in the embodiment described herein. System 300 is representative of processing systems based on the PENTIUM.RTM. Pro, PENTIUM.RTM. II, PENTIUM.RTM. III, Itanium.RTM. microprocessors available from Intel Corporation of Santa Clara, Calif., although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 300 may be executing a version of the WINDOWS.TM. operating system available from Microsoft Corporation of Redmond, Wash., although other operating systems and graphical user interfaces, for example, may also be used. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.

Detailed Description Text (8):
System 300 includes a memory 320. Memory 320 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 320 may store instructions and/or data represented by data signals that may be executed by processor 302. A cache memory 304 can reside inside processor 302 that stores data signals stored in memory 320. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (9):
An impedance control mechanism 306 for a center tapped termination bus also resides in memory controller hub 316. Alternate embodiments of an impedance control mechanism 306 can also be used in microcontrollers, embedded processors, graphics devices, DSPs, and other types of logic circuits.

Detailed Description Text (10):
A system logic chip 316 is coupled to the processor bus 310 and memory 320. The system logic chip 316 in the illustrated embodiment is a memory controller hub (MCH). The processor 302 communicates to the MCH 316 via a processor bus 310. The MCH 316 provides a high bandwidth memory path 318 to memory 320 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 316 directs data signals between processor 302, memory 320, and other components in the system 300 and bridges the data signals between processor bus 310, memory 320, and system I/O 322. In some embodiments, the system logic chip 316 provides a graphics port for coupling to a graphics controller 312. The MCH 316 is coupled to memory 320 through a memory interface 318. The graphics card 312 is coupled to the MCH 316 through an Accelerated Graphics Port (AGP) interconnect 314.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments KWIC | Draw Desc | Image

☐ 22. Document ID: US 6347351 B1

L3: Entry 22 of 23 File: USPT Feb 12, 2002

DOCUMENT-IDENTIFIER: US 6347351 B1
TITLE: Method and apparatus for supporting multi-clock propagation in a computer system having a point to point half duplex interconnect

Brief Summary Text (5):
A hub interface is an input/output (I/O) interconnect for connecting I/O hubs and Peripheral Component Interconnect (PCI) bridges adhering to a Specification Revision 2.1 bus developed by the PCI Special Interest Group of Portland, Oreg., to a memory controller hub. A Hub interface is a half-duplex bus with a distributed arbiter. Synchronization on the interface occurs by a global clock and two request (REQ) signals. Agents at each side of the interface assert REQ signals to convey a request to the other agent. For example, one agent (e.g., side A) sends a request signal to the other agent (e.g., side B), while side B transmits a request to side A. The REQ signals are sampled at each side, and a decision is made as to which side will be granted access of the interface.

Detailed Description Text (5):
MCH 110 may also include a graphics interface 113 coupled to a graphics accelerator 130. In one embodiment, graphics interface 113 is coupled to graphics accelerator 130 via an accelerated graphics port (AGP) that operates according to a Specification Revision 2.0 interface developed by Intel Corporation of Santa Clara, Calif. In addition, MCH 110 includes a hub interface controller 120. Interface controller 120 is used to couple MCH 110 to an input/output control hub (ICH) 140 via a hub interface A. ICH 140 provides an interface to input/output (I/O) devices within computer system 100. ICH 140 also includes a hub interface controller 120 that is used for coupling to MCH 110.

Detailed Description Text (8):
ICH 140 may also include a PCI bridge 146 that provides an interface to a PCI bus. PCI bridge 146 provides a data path between CPU 102 and peripheral devices. Devices that may be coupled to PCI bus 142 include an audio device 150 and a disk drive 155. However, one of ordinary skill in the art will appreciate that other devices may be coupled to PCI bus 142. In addition, one of ordinary skill in the art will recognize that CPU 102 and MCH 110 could be combined to form a single chip. Further graphics accelerator 130 may be included within MCH 110 in other embodiments.

Detailed Description Text (42):
In alternative embodiments, additional transaction attributes may include the ability to differentiate between "snooped" traffic where cache coherency is enforced by hardware (i.e., chipset) and "non-snooped" traffic that relies on software mechanisms to ensure data coherency in the system. Moreover, another possible attribute would be an "explicitly prefetchable" hint, to support a form of read caching and allow for more efficient use of the main memory bandwidth.

☐ 23. Document ID: US 6292048 B1

L3: Entry 23 of 23 File: USPT Sep 18, 2001

DOCUMENT-IDENTIFIER: US 6292048 B1
TITLE: Gate enhancement charge pump for low voltage power supply

Detailed Description Text (10):
System 300 includes a memory 320. Memory 320 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. A cache memory 304 can reside inside processor 302 that stores data signals stored in memory 320. Alternatively, in another embodiment, the cache memory may reside external to the processor.

Detailed Description Text (11):
A system logic chip 316 is coupled to the processor bus 310 and memory 320. The processor 302 communicates to a memory controller hub (MCH) 316 via a processor bus 310. The MCH 316 provides a high bandwidth memory path 318 to memory 320 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 316 directs data signals between processor 302, memory 320, and other components in the system 300 and bridges the data signals between processor bus 310,

memory 320, and system I/O 322. The graphics card 312 is coupled to the MCH 316 through an Accelerated Graphics Port (AGP) interconnect 314.

Detailed Description Text (13):
For another embodiment of a system, one implementation of a charge pump can be used with a system on a chip. One embodiment of a system on a chip comprises of a processor and a memory. The memory for one system is a flash memory. The flash memory can be located on the same die as the processor and other system components. Additionally, other logic blocks such as a memory controller or graphics controller can also be located on a system on a chip. By including one embodiment of the present invention on the system on a chip, the flash memory can be enabled to program and erase flash memory cells without requiring a high voltage pin on the system on a chip pin-out. The needed high voltage potentials can be generated on the same die.

Terms	Documents
L2 and graphic$3	23